Theory Refinement of Bayesian Networks with Hidden Variables

by

Sowmya Ramachandran, B.Tech., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 1998

Theory Refinement of Bayesian Networks with Hidden Variables

Approved by
Dissertation Committee:

To my grandfather

Acknowledgments

Getting a doctorate is an exciting, challenging, and at times difficult process. I have been fortunate to have an extensive network of family, friends, and professional colleagues to support me throughout this process. I am grateful to Ray Mooney for supervising my research efforts, and all the invaluable advice he has given me through the years on technical matters and on presentation skills. I am also grateful to him for supporting me through the following grants: NSF grant number IRI-9704943, and NSF grant number IRI-9310819. My research was also supported by teaching assistantships in the Department of Computer Sciences. I would also like to thank Ben Kuipers for financial support and advice on what research is all about. I would like to thank Greg Provan, Risto Miikkulainen, Joydeep Ghosh, and Ben Kuipers for serving on my committee and giving me insightful feedback on my research. I am also grateful to Haym Hirsh for all the support and encouragement through the years.

I am very grateful to Bobby Blumofe, Lorenzo Alvisi, Mike Dahlin, Calvin Lin and other folks at the LESS lab for letting me use their computing facilities for research. The multiprocessor computing facilities in this lab were made available through a generous equipment donation from Sun Microsystems. But for these machines, I would still be a graduate student.

My tenure as a graduate student was enlivened and enriched by all my fellow graduate students, who have shown me that there is more to life than graduate school. I would like to thank James Lester, Charles Callaway, Tim Collins, and Bradley Richards for all the juggling and dessert, to Jeff Rickel, and Siddarth Subramanian for enlivening our graduate student lives with pranks and spiders. It has been very nice sharing an office with Cindi Thompson and Mike Hewett, and knowing that I can always count on them for support, friendship, and proof-reading. Thanks also to Marty and Coquis Mayberry for Pictionary, Scrabble, and several interesting discussions. I am also very grateful to Ajita John, Gita Mani, Ashok Balivada, and Tom Lee for being such good friends. Other students who have supported in my research include Jeff Mahoney, John Zelle, Paul Baffes, Rich Mallory, Tara Estlin, Dave Moriarty, Mary Elaine Califf, Shailesh Kumar and many others. I would also like to take this opportunity to remember Bert Kay, who passed away recently, and his contributions to my life as a graduate student. Bert was a wonderful role model, a patient teacher, and a very kind friend. I am very grateful to have had a chance to know him. I am especially thankful to Rick Froom for standing by me these last few years, for showing me a lighter and brighter side of life, and for all the support and encouragement.

Finally, I would like to thank my parents and grandparents for their patience and understanding, and for providing me with a place to call home. Without their love and support, I would not be where I am today. Finally, I would like to thank my sister and her two beautiful children for making me laugh and reminding me that life is delightful.

Sowmya Ramachandran

The University of Texas at Austin
May 1998

Theory Refinement of Bayesian Networks with Hidden Variables

Publication No. ______________

Sowmya Ramachandran, Ph.D.
The University of Texas at Austin, 1998

Supervisor: Raymond J. Mooney

Research in theory refinement has shown that biasing a learner with initial, approximately correct knowledge produces more accurate results than learning from data alone. While techniques have been developed to revise logical and connectionist representations, little has been done to revise probabilistic representations.

Bayesian networks are well-established as a sound formalism for representing and reasoning with probabilistic knowledge, and are widely used. There has been a growing interest in the problem of learning Bayesian networks from data. However, there is no existing technique for learning or revising Bayesian networks with hidden variables (i.e. variables not represented in the data), that has been shown to be efficient, effective, and scalable through evaluation on real data. The few techniques that exist for revising such networks perform a blind search through a large space of revisions, and are therefore computationally expensive.

This dissertation presents BANNER, a technique for using data to revise a given Bayesian network with Noisy-Or and Noisy-And nodes, to improve its classification accuracy. Additionally, the initial network can be derived directly from a logical theory expressed as propositional Horn-clause rules. BANNER can revise networks with hidden variables, and add hidden variables when necessary. Unlike previous approaches to this problem, BANNER employs mechanisms similar to those used in logical theory refinement techniques for using the data to focus the search for effective modifications to the network. It can also be used to learn networks with hidden variables from data alone. We also introduce BANNER-PR, a technique for revising the parameters of a Bayesian network with Noisy-Or/And nodes, that directly exploits the computational efficiency afforded by these models.

Experiments on several real-world learning problems in domains such as molecular biology and intelligent tutoring systems demonstrate that BANNER can effectively and efficiently revise networks to significantly improve their accuracies, and thus learn highly accurate classifiers. Comparisons with the Naive Bayes algorithm show that using the theory refinement approach gives BANNER a substantial edge over learning from data alone. We also show that BANNER-PR converges faster and produces more accurate classifiers than an existing algorithm for learning the parameters of a network.

Contents

Chapter 1

Introduction

Theory refinement, or theory revision, is an area of research that has grown out of work on inductive and explanation-based learning. It is based on the idea that, when only limited data is available, biasing an inductive learner with prior knowledge can improve learning by focusing the search space of possible hypotheses. The more complex the domain, the more the advantage of such a bias. Thus, theory refinement systems assume that the learner has an initial imperfect knowledge base (usually obtained from a domain expert) which is then inductively revised to fit the data. Many techniques have been developed for revising knowledge bases represented in various languages such as propositional Horn-clause logic (Ourston & Mooney, 1994; Koppel, Feldman, & Segre, 1994) and first-order Horn-clause logic (Cohen, 1992; Wogulis & Pazzani, 1993; Wogulis, 1994; Richards & Mooney, 1995; Brunk, 1996). Even within the connectionist framework there are techniques, such as KBANN (Towell & Shavlik, 1994; Opitz & Shavlik, 1993), that explicitly bias a neural network with an initial theory. Experiments on real-world data have demonstrated that revising an approximate domain theory produces more accurate results than learning from training data alone (Ourston & Mooney, 1990; Thompson, Langley, & Iba, 1991; Towell, Shavlik, & Noordewier, 1990).

Although existing techniques for revising theories differ in their approaches, they have many things in common. One such common feature is that most of them are geared towards learning theories for the specific task of classification. They use the classification accuracy of the theory as an indication of the correctness of the current theory, and make revisions to a theory only if it misclassifies some of the data. A second, very important feature that they share in common is that, rather than searching blindly through the entire space of possible revisions, they analyse the classification errors resulting from the theory, and use this information to restrict the space of possible revisions to be considered. These features combine to make these algorithms directed and, therefore, efficient.

Research in theory refinement has, however, been limited to logical and connectionist representations. Little has been done to address the problem of revising probabilistic knowledge. Intelligent systems need mechanisms to represent and reason with uncertain knowledge. Uncertainty in a domain or a task may arise due to incomplete knowledge about the state of the world or due to true randomness in the domain. Examples of tasks involving uncertainty include medical diagnosis, and plan recognition, to name but a few. Thus, we need languages for representing uncertainty. It is also desirable to have techniques for learning and revising knowledge represented in these languages.

Several approaches have been developed for representing and reasoning with uncertainty. These include *default reasoning* (Reiter, 1980), *fuzzy logic* (Zadeh, 1965, 1981), *certainty factors* (Buchanan & Shortliffe, 1984), *Bayesian networks* (Pearl, 1988), and *Dempster-Shafer* calculus (Dempster, 1968; Shafer, 1976). Of these, default reasoning uses a symbolic representation of uncertainty, while the rest use numeric representations. The Bayesian network approach stands out as the only one that is directly grounded in probability theory, which has long been a widely accepted formalism for representing uncertainty. The rest of the approaches invent their own calculi, which makes their semantics less clearly defined.

Among the numeric representations, *certainty factors* have played a significant role in the history of uncertain reasoning. The use of *certainty factors* is exemplified in MYCIN, an expert system to recommend treatment for bacterial infection (Shortliffe & Buchanan, 1975; Buchanan & Shortliffe, 1984). MYCIN is a rule-based system, where the rules are augmented with *certainty factors* or numbers that indicate their credibility. The certainty of a conclusion of a rule is computed as a function of the certainty of the premises and the credibility of the rule. Although they have been successfully applied in a number of domains, the rules for combining certainty factors are *ad hoc* and provide no mathematical guarantees, unless unrealistic independence assumptions are made (Heckerman, 1986).

Bayesian networks (Pearl, 1988) represent uncertainties as probabilities of events in the world, and provide ways for representing dependencies between variables explicitly. A Bayesian network is a directed acyclic graph (DAG), where the nodes are random variables representing events, and the links represent dependencies between the variables. Associated with the DAG are numeric parameters that define the dependencies precisely. A variety of techniques for reasoning with such networks have been developed that use theoretically sound mechanisms for combining probabilities, and accounting for dependencies. Their strong grounding in probability theory makes Bayesian networks a particularly attractive formalism for representing knowledge. Many real-world applications, especially in medical diagnosis, now use this representation (Pradhan, Provan, Middleton, & Henrion, 1994; Burnell & Horovitz, 1995; Fung & Del Favero, 1995). However, like all numerical representation schemes, Bayesian networks suffer from the knowledge acquisition problem. Not only is it difficult to formulate the underlying structure of the DAG, it is especially difficult to specify the dependencies between the variables in precise numeric terms. Therefore, it would be useful to have efficient techniques that learn such networks from data. Not surprisingly, there is now a growing interest in this problem.

The task of learning a Bayesian network can be divided into two subtasks: one of learning the DAG structure of the network, and the second of determining the parameters. Within the general framework of inducing Bayesian networks, we can envision the following scenarios.

1. *Known structure, complete data*: In this scenario, the structure of the network is given and assumed to be correct and the data includes observations of all the variables in the network. The task here is to learn the parameters of the network from data.

2. *Known structure, incomplete data*: Here, the structure of the network is given and assumed to be correct. The data, however, does not include observations of every variable in the network. This includes situations where some variables are missing values for part of the data, and

situations where the values of some variables are never specified in the data. Variables whose values are never included in the data are called *hidden variables*. Again, the task is to induce the parameters of the network. However, the task is more complicated in this case due to the presence of hidden variables and variables with missing values.

3. *Unknown structure, complete data*: In this case, neither the structure, nor the parameters of the network are known. We can, however, assume that the data is complete, and there are no hidden variables. The task here is to learn both the structure and the parameters of the network.

4. *Unknown structure, incomplete data*: This is the most general learning scenario where the structure of the network is unknown and there are hidden variables.

The first of these is fairly straightforward. A common approach is to use the maximum likelihood estimates for the parameters, which in the case of complete reduces to a function of the relative frequencies of occurrences of the values of the variable (Spiegelhalter & Lauritzen, 1990).

The problem of learning the parameters for a network with a known structure, given incomplete data, has also received some attention. Many statistical techniques like *Gibbs sampling* (Geman & Geman, 1984) and *EM* (Dempster, Laird, & Rubin, 1977; Lauritzen, 1995) can be used in the context of Bayesian networks. Russell, Binder, Koller, and Kanazawa (1995) have proposed an approach that attempts to optimize the probability of the data given the network using a gradient descent algorithm.

The learning problem addressed by Cooper and Herskovits (1992) falls in the third category. Their technique, implemented in a system called K2, uses a scoring metric to hill climb through a space of possible Bayesian networks to find one that is the most probable given the data. A number of variations and improvements to this approach have since been proposed (Buntine, 1991; Heckerman, Geiger, & Chikering, 1994; Provan & Singh, 1994).

The fourth scenario above, namely that of learning a Bayesian network with an unknown structure, given incomplete data, is by far the most difficult, especially when there are hidden variables. Most of the above techniques could be adapted to discover hidden variables, but at a great cost involving brute force search. Connolly (1993) has proposed using clustering techniques (Fisher, 1987) to discover hidden variables. However, this technique can only learn tree-structured networks. Very recently, Friedman (1997) has proposed a technique, called MS-EM, that extends EM (Dempster et al., 1977; Lauritzen, 1995) to learn the structure of a network with hidden variables. However, it is not flexible because it requires that the number of hidden variables to be considered by the network be specified ahead of time. BKD (Ramoni & Sebastiani, 1997) is another recent algorithm, based on K2, for learning Bayesian networks from incomplete data. However, it cannot discover hidden variables.

Thus, while researchers have a grasp on some aspects of learning Bayesian networks, the problem of learning Bayesian networks with unknown structures and hidden variables still poses a tough challenge. However, theory refinement techniques like those proposed by Ourston and Mooney (1994), Opitz and Shavlik (1993), Mahoney and Mooney (1993) have been successful in addressing similar issues. The bias that such techniques provide in the form of prior knowledge, and their techniques for using the data to focus the search for accurate theories, make them very efficient. We had mentioned earlier that there has been very little research into using theory refinement techniques to learn Bayesian networks. Lam and Bacchus (1994b) have a technique for incrementally refining a Bayesian network using the Minimum Description Length principle (Rissanen, 1978). Buntine (1991) has proposed a technique for revising a Bayesian network efficiently, using scoring metrics similar to that proposed by (Cooper & Herskovits, 1992). However, neither of these techniques can revise networks with hidden variables. While algorithms like K2 (Cooper & Herskovits, 1992), BKD (Ramoni & Sebastiani, 1997) and MS-EM (Friedman, 1997), that build a Bayesian network iteratively, can also be used for theory refinement, they perform a blind search through all possible single revisions to the network, and are thus computationally expensive. Mahoney and Mooney (1993) have a system for revising uncertain knowledge bases, but expressed in the form of *certainty factors*. Their system, RAPTURE maps the rules of the knowledge base into a neural network and uses connectionist methods to revise the certainty factors associated with the rules as well as the rules themselves. RAPTURE can also discover hidden variables not specified in the data.

All of the Bayesian network learning techniques described above optimize the probability assigned by the network to the data as a whole. However, in many learning situations the objective is to acquire knowledge for a particular task such as classification. In such situations, it is better to learn theories that are optimized for the particular task that it is meant to perform. Thus, it is desirable to have a learning algorithm that optimizes the *classification* accuracy of the network, when the objective is to build a Bayesian network for the purpose of classification. Friedman and Goldszmidt (1996) have experimentally demonstrated the advantage of learning to maximize classification accuracy. Specifically, the learning algorithm should try to learn a network that best estimates the probability distribution of the class variables conditioned on some evidence variables. Currently, there are very few existing techniques for learning Bayesian networks that optimize for classification.

The goal of our research is to use a theory refinement approach to learning Bayesian network classifiers with hidden variables. From the perspective of the four learning scenarios outlined earlier, this problem lies somewhere between scenarios 2 and 4. Thus, we assume that the learner is given a Bayesian network that may be incomplete or incorrect. We also assume that the learner is provided with data that may not include all of the variables in the network. The task is to use the data to improve the predictive accuracy of the network, modifying both its parameters and structure (adding hidden variables when needed). Unlike the previous approaches to revising Bayesian networks, we are specifically interested in developing mechanisms, similar to those employed by logical theory refinement techniques (Ourston & Mooney, 1994; Koppel et al., 1994; Cohen, 1992; Wogulis & Pazzani, 1993; Richards & Mooney, 1995; Brunk, 1996), for using the data to focus the search for effective modifications to the network.

Since general Bayesian networks are impractical for many large problems because the size of the conditional probability tables grows exponentially in the fan-in of a node, we focus on the problem of learning networks with Noisy-Or and Noisy-And nodes (Pearl, 1988; Pradhan et al., 1994). These are specialized models for representing dependencies that only require a linear number of parameters. We specifically chose these models because they are semantically close to *logical ors* and *logical ands*, thus making it possible to use knowledge expressed in the form of logical rules as an initial theory. Using such nodes, a knowledge base originally expressed as rules can be mapped to an analogous Bayesian network and refined to improve its accuracy. Many existing knowledge bases are written in the form of rules, and many experts have become comfortable with this formalism. However, results in theory refinement show that the accuracy of such rule bases can be dramatically improved by mapping them to a representation that employs some form of uncertain reasoning or numerical summing of evidence (Towell & Shavlik, 1994; Koppel et al., 1994; Baffes & Mooney, 1993; Mahoney & Mooney, 1993).

Here, we present a system called BANNER that, given a set of data and an initial approximate domain theory, produces a revised theory in the form of a Bayesian network with Noisy-Or/And nodes that accurately classifies the data. The initial theory may be in the form of propositional Horn-clause rules, or in the form of a Bayesian network with Noisy-Or/And nodes. Rather than search through the space of all possible revisions, it uses information in the data to select specific nodes and links in the network to be revised, which makes it efficient. Although designed for theory refinement, it can also be used to learn Bayesian networks with hidden variables inductively from data using a default initial network.

BANNER uses a two-tiered approach, similar to that used by RAPTURE (Mahoney & Mooney, 1993, 1994). Given some data, BANNER first tries to improve the Bayesian model by revising the parameters of the network. If the network still does not fit the training data, the structure of the network is modified to find the network with the highest predictive accuracy. Thus, BANNER has two components: one for revising the parameters of the networks, and one to revise the structure.

In the following chapters, we will present the details of our technique, and provide experimental evaluations of its effectiveness. We will evaluate our system on the following real-world learning problems, some of which are standard benchmark problems used to evaluate theory refinement and inductive learning techniques: recognizing DNA promoters (Noordewier, Towell, & Shavlik, 1991), recognizing DNA splice-junctions (Noordewier et al., 1991), learning student models for a C++ tutor (Baffes, 1994), diagnosing brain disorders in human patients (Tuhrim, Reggia, & Goodall, 1991), and predicting the outcome of a chess end-game from board configurations (Shapiro, 1983, 1987). The first four of these problems have associated domain theories, represented as propositional Horn-clause rules, that do not have good predictive accuracies on the data and, therefore, need to be revised. The fifth problem does not have an initial domain theory and is intended to study the effectiveness of BANNER in learning networks from scratch. In addition, we will evaluate the structure revision component on BANNER by performing experiments on corrupted versions of the theory for recognizing DNA promoters. Through our experiments, we will demonstrate that our technique is effective in learning fairly large Bayesian networks with high classification accuracy. We will show that the performance of BANNER on these domains is comparable to the best results obtained with other learning techniques. We will also show that the strategy of starting with an initial, approximately correct theory gives BANNER an edge over systems that learn from scratch. The experiments with corrupted theories will demonstrate the effectiveness of our technique in revising such theories so as to significantly improve their accuracies.

The research presented here makes contributions to the field of Bayesian network learning as well as the field of theory refinement. From a Bayesian network learning perspective, BANNER is a novel technique for revising a class of Bayesian network classifiers with hidden variables that can also add hidden variables when necessary. It can also be used to learn networks with hidden variables inductively from incomplete data. Our experiments on real-world data sets demonstrate that this approach can efficiently revise large networks and produce highly accurate classifiers. Whereas previous techniques for revising Bayesian networks searched through the space of all possible revisions, our technique uses novel mechanisms for using the information in the data to guide the search for useful revisions, thus eliminating a large number of irrelevant revisions from consideration. Since the initial theory given to BANNER may be in the form of propositional Horn-clause rules, it also provides a direct mechanism for incorporating knowledge expressed as propositional Horn-clause rules into a Bayesian network. We also introduce a new technique for revising the parameters of a network with Noisy-Or/And nodes that directly exploits the efficiency afforded by these models, and is targeted towards learning classifiers by trying to optimize the conditional distribution of the class variables given the evidence. We show that this technique converges faster and produces more accurate classifiers than an existing algorithm for learning the parameters of a network.

From a theory refinement perspective, this dissertation presents a novel hybrid theory refinement system that combines good performance with comprehensibility. Early research in theory refinement focussed on purely symbolic representations, which are very comprehensible, but show poor accuracies on several real-world learning problems. Techniques that revise rules-bases by mapping them into representations that combine logical rules and some form of uncertain reasoning, or numerical summing of evidence, have been shown to produce significantly improved classifiers for many domains. Successful hybrid theory refinement systems include KBANN, a system for learning a class of multi-layered feedforward neural networks called *Knowledge-Based neural networks*, whose structures are determined by an initial logical theory (Towell & Shavlik, 1994), and RAPTURE, a system that revises logical theories by mapping then into *certainty factors* rule-bases (Mahoney & Mooney, 1993; Buchanan & Shortliffe, 1984). Although these techniques have been shown to learn highly accurate classifiers, such hybrid representations lack well-defined semantics and cannot be easily understood. Bayesian networks, on the other hand, are attractive as hybrid representations because they combine sound mechanisms for representing probabilities with a well-founded qualitative representation of the correlations between variables, and are grounded in probability theory. Several sound inference mechanisms have been developed to reason with such networks. Thus, BANNER can be viewed as a hybrid theory refinement system that learns representations with more clearly defined semantics than the representations learned by the other hybrid theory refinement systems mentioned above, while maintaining a comparable level of performance in terms of learning accurate classifiers, as will be demonstrated by our experiments.

This dissertation is organized as follows: Chapter 2 presents a brief introduction to various concepts that are central to our research, Chapter 3 presents an overview of BANNER, Chapter 3 discusses the parameter revision component, Chapter 4 discusses the structure revision component, Chapter 6 presents experiments that demonstrate the effectiveness of BANNER, Chapters 7 and 8 discuss related work and future directions respectively, and finally Chapter 9 summarizes our research and its contributions.

Chapter 2

Background

Our research builds upon several ideas and techniques. This chapter presents some of this background knowledge at a level of detail that will contribute to an easier understanding of later chapters. We will begin with an overview of *Bayesian networks*. Then we will look at *multi-layered feedforward neural networks*, followed by an overview of some techniques for *theory refinement*, and techniques for *learning Bayesian networks*.

2.1 Bayesian Networks: An Overview

Bayesian networks (Pearl, 1988) provide a formalism for representing probabilistic knowledge. They provide theoretically sound mechanisms for representing and reasoning with probabilistic dependencies between events. In general, a Bayesian network is a directed acyclic graph, whose nodes represent to random variables. Here, we use the convention of using upper-case letters to represent variables and lower-case letters to represent values associated with the variable. The links in the network represent dependencies between variables, such that two variables are assumed to be independent of each other if there is no undirected path between them, or if any of their common ancestors are instantiated. Associated with each node is a *conditional probability table (CPT)*, which gives the probability of each value of the variable given each possible combination of values of its parent nodes. The CPTs define how the influences of the parents of a node interact to produce a combined influence on the node. Given a network with n nodes and the associated CPTs, the probability of a conjunction of a particular assignment of values to the variables, i.e. $P(x_1, \ldots, x_n)$, can be calculated using the following formula:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(X_i)) \tag{2.1}$$

where $P(x_i \mid Parents(X_i))$ is obtained from the CPT associated with variable X.

Figure 2.1 (Pearl, 1988) shows an example of a Bayesian network. Such a network maybe used by a person to decide whether or not to respond to an alarm in his house. The nodes in the network represents the various events that are of relevance to the decision. Node E represents the occurrence of an earthquake, node R represents the announcement of an earthquake on the radio, and node D represents the event of the person's daughter calling him about the alarm. The links between the nodes represent the dependencies between the various events. For instance, the network indicates that A is conditionally dependent on B and E, while G is independent of B given A. The links can also be seen to indicate *causality.* Thus, the link from *earthquake* to *alarm* can be interpreted as a statement that earthquake *causes* the alarm to go off. The CPT associated with each variable reflects the interaction between the causes directly influencing a variable, and the strength of the influences. For example, the CPT associated with variable A in Figure 2.1 would specify:

$$M(A) = \begin{bmatrix} P(A \mid \neg B, \neg E) & P(A \mid B, \neg E) & P(A \mid \neg B, E) & P(A \mid B, E) \\ P(\neg A \mid \neg B, \neg E) & P(\neg A \mid B, \neg E) & P(\neg A \mid \neg B, E) & P(\neg A \mid B, E) \end{bmatrix} \tag{2.2}$$

Given the CPTs, the joint probability distribution of the variables in the network can be computed as follows:

$$P(b, e, r, a, d, w, g) = P(b)P(e)P(r \mid e)P(a \mid e, b)P(d \mid a)P(w \mid a)P(g \mid a) \tag{2.3}$$

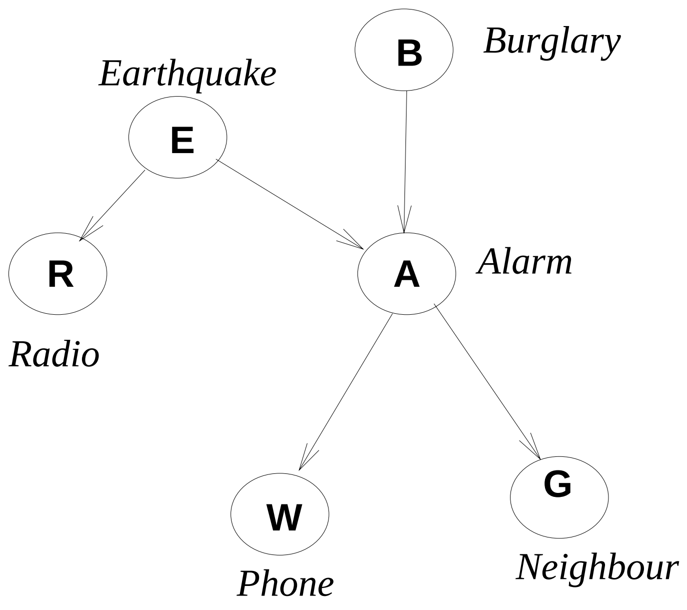


Figure 2.1: Bayesian Network - Example

Typically, a Bayesian network is used to infer the probability distribution of a set of variables T, given the value of another set of variables E (called *evidence*). This inference is sometimes *causal* (i.e. inferring effect from causes), as in the case where one uses the network in Figure 2.1 to infer the probability that his daughter will call, given evidence that a burglary has occurred. Sometimes, the inference is *diagnostic* or *evidential* (i.e. inferring causes from effects), as in the case when one uses the network to infer the probability of a burglary, given the evidence that his daughter called him to report the alarm. It is also possible to combine causal and diagnostic inferences. For instance, one might want to infer the probability of the alarm going off, given the evidence that no burglary was known to occur, but that the radio announced an earthquake.

2.1.1 Polytree Vs. Loops

The structure of a Bayesian network has a significant influence on the complexity of inference. Based on their structure, Bayesian networks can be divided into two classes: *polytrees*, which have a simple structure where each pair of variables is connected by at most one path, and *networks with loops*, which have undirected cycles in the structure. The inference algorithm for polytrees is polynomial in the size of the network and the propagation rules for beliefs are local (Pearl, 1988).

The derivation of the propagation rules for polytrees exploit the fact that the network is singly connected and cannot be used in the presence of loops. Several algorithms have been proposed for handling loops (Pearl, 1988). *Clustering* is a technique where the variables forming a loop are clustered into one node, which results in a polytree. Inference is done using the polytree algorithm on the clustered network. The clustered nodes are then separated into individual nodes whose beliefs are computed from the belief of the cluster. *Conditioning* is a technique which uses case analysis to propagate beliefs. The algorithm picks a variable from each loop and this set of variables is instantiated to all the possible values that the variables can take. Instantiating a variable in each loop breaks the loops and the network reduces to a polytree. Beliefs are propagated for each of these cases using the polytree algorithm. The overall belief of each node is computed as the weighted average of the belief computed for the node for each of the cases. *Stochastic simulation* is an algorithm for simulating the network starting at some random state. The belief associated with each variable is the frequency of occurrence of each value of the node over a large number of simulations. It has been shown that, in general, inference in the presence of loops is NP-hard (Cooper, 1990).

As mentioned earlier, Bayesian networks are used to infer the probabilities of some variable given the value of some other set of variables as evidence. For a network with loops, if all the variables that are needed to break the loops are always included in the set of evidence variables, then the network may be treated as a polytree for purposes of inference. Henceforth, we will refer to such networks as *virtual polytrees.* For the loop shown in Figure 2.2, instantiating variables A, B, or C will break loop, whereas instantiating variable D will not. Figure 2.3 shows a Bayesian network with three undirected loops. The first loop is formed by variables A, C, D, and E, the second loop is formed by the variables B, C, D and E, and the third loop is formed by the variables A, B, and C, and D. For this network, instantiating the values of variables A and B breaks all the loops in the network and makes the network a virtual polytree.

2.1.2 Noisy-Or and Noisy-And Models

The specification of a Bayesian network, in its most general form, is combinatorial in the fan-in of the nodes. It requires the specification, for each variable, of the conditional probabilities of the variable given all possible combinations of values of its parents. For a network where all variables are binary-valued, a variable with n parents would require 2^n conditional probabilities to be specified. This makes inference exponential in the fan-in of the nodes. In addition, learning such a network requires the estimation of an exponential number of parameters and therefore requires a lot of data. The need for an exponential number of parameters is a result of allowing for specifications of arbitrary interactions between the influences due to the parents of a node. Henceforth, we will refer to nodes modeling such unrestricted interactions as *general* nodes. The *Noisy-Or* and the

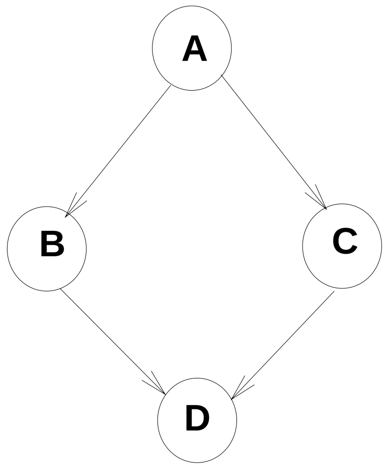


Figure 2.2: Example of a Loop

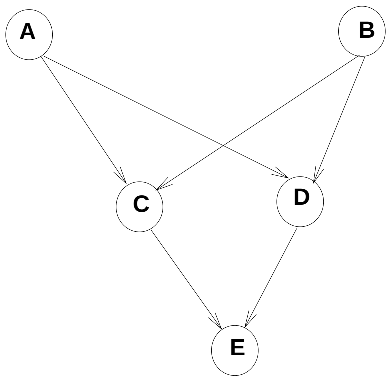


Figure 2.3: Example of a Network with Loops

Noisy-And nodes (Pearl, 1988) avoid the problem of exponential parameters by defining a more structured model of the interactions between the parents of a node, so that the combined influence of the parents can be computed from the influence of each parent in isolation. As a consequence of imposing such a structure, the number of parameters associated with networks consisting of such nodes is linear in the number of links in the network. Using these parameters for inference significantly reduces storage space and processing time. Learning such networks is also easier since only a linear number of parameters need to be estimated. Thus, the number of parameters associated with the network, and hence the numbers of parameters to be estimated while learning such networks, is linear in the number of links in the network. In the following subsections, we further elaborate on the specific combination rules provided by each of these models.

The Noisy-Or model

A *Noisy-Or* node in a Bayesian network is a generalization of a logical *or*. As in the case of the logical *or*, an event, represented by a Noisy-Or node N_j is presumed to be false (i.e. $P(N_j = true) = 0$) if all the conditions that cause N_j are false. However, unlike a logical *or*, if one of the causes of the event N_j is true, it does not necessarily imply that N_j is definitely true. Each condition N_i causing the event N_j can be thought of as having an associated inhibitory influence which is active with a probability q_{ij}. Thus, if N_i is the only cause of N_j that is true, then N_j is true with a probability $(1 - q_{ij})$. Moreover, the inhibitory influences are assume to be mutually independent. The likelihood of N_j being true is a monotonic function of the number of its causal conditions that are true. The parameter $c_{ij} = 1 - q_{ij}$ is the degree to which an isolated cause N_i of an event N_j can endorse the event. These parameters can be used to construct a conditional probability table for the node, if needed. Figure 2.4 shows the conceptual view of a Noisy-Or node.

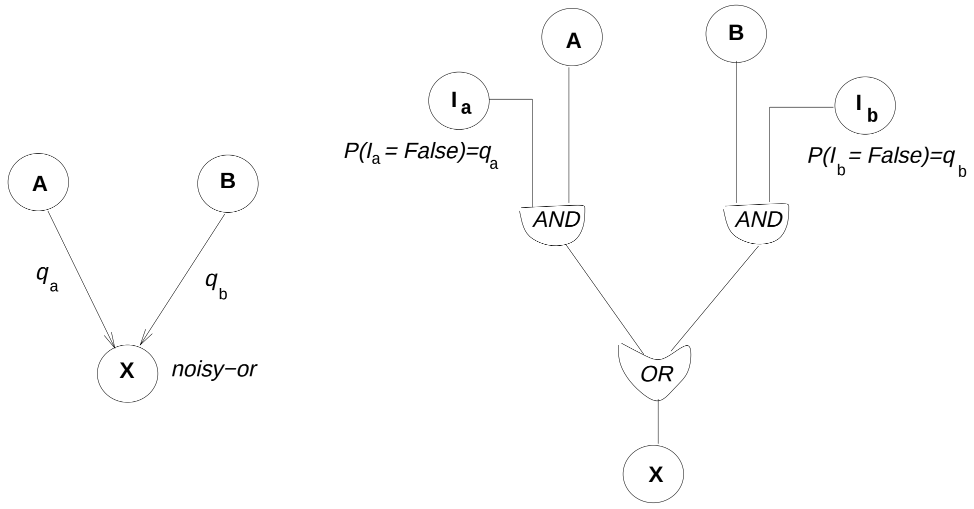


Figure 2.4: Conceptual View of Noisy-Or Node

Given some evidence, the parameters associated with each link in a network, and the belief measures of all the parents of a node in the network, there is a simple equation for calculating the *degree of belief* that the node is true. Under the assumption that all the evidence in the network is causally upstream of the node, and that the network is a virtual polytree, the degree of belief in a node N_j is given by

$$Bel(N_j = x) = \begin{cases} \prod_i (1 - c_{ij} Bel(N_i = True) & \text{if } x = False \\ 1 - \prod_i (1 - c_{ij} Bel(N_i = True) & \text{if } x = True \end{cases} \quad (2.4)$$

where c_{ij} is the parameter on the link from node N_i to node N_j. Note that the above computation is linear in the fan-in of the node.

The Noisy-And model

A *Noisy-And* node is the dual of a Noisy-Or node. It is a generalization of a logical *and*. As in the case of a logical *and*, an event represented by a Noisy-And node N_j is presumed to be true if all the conditions that cause N_j are true (i.e. $P(N_j = True) = 1$). However, unlike the logical *and*, if one of the causes of the event N_j is false, it does not imply that N_j is definitely false. Each condition N_i causing N_j can be thought of as having an associated enabling influence which is active with a probability q_{ij}. Thus, if N_i is the only cause of N_j that is false, then N_j is false with a probability $(1 - q_{ij})$. Moreover, the enabling influences are assumed to be mutually independent. The likelihood of N_j being false is a monotonic function of the number of its causes that are false. The parameter c_{ij} is the degree to which disproving an isolated cause of an event disproves the event itself. Figure 2.5 shows the conceptual view of a *Noisy-And* node.

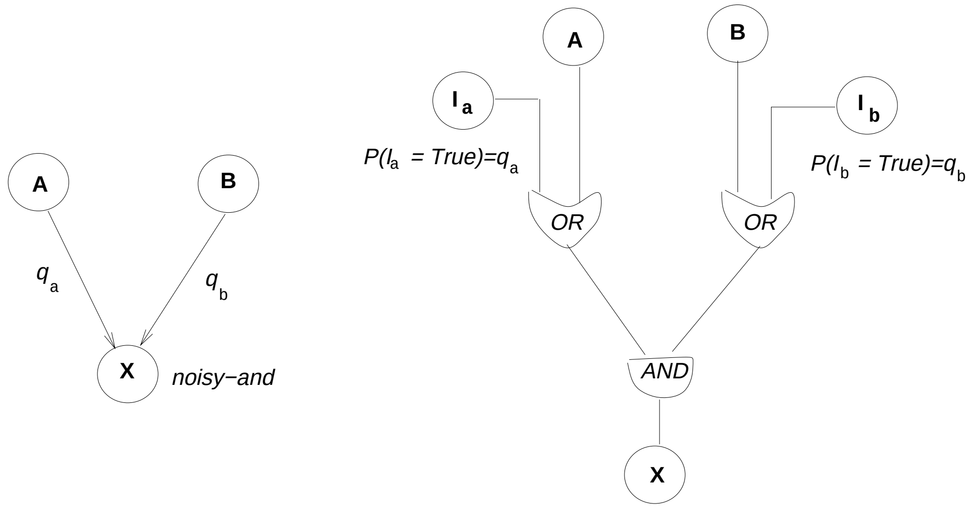


Figure 2.5: Conceptual View of Noisy-And Node

The belief measure of a Noisy-And node N_j, given some evidence, the parameters associated with each link in the network, and the belief measures of all the parents of the node, is given by

$$Bel(N_j = x) = \begin{cases} 1 - \prod_i (1 - c_{ij}(1 - Bel(N_i = True))) & \text{if } x = False \\ \prod_i (1 - c_{ij}(1 - Bel(N_i = True))) & \text{if x = True} \end{cases} \quad (2.5)$$

where c_{ij} is the parameter on the link from node N_i to node N_j. Here again, the assumption is that all the evidence in the network is causally upstream of the node, and that the network is a virtual polytree.

Leak Nodes

The Noisy-Or and Noisy-And models described above assume that all the possible causes of a particular event are known. For example, a Noisy-Or node is specified to be definitely false when all of its parents are false. The reasoning is that, if none of the causes of an event modeled by the Noisy-Or node are true, then the event itself can not be true. This assumption that all the possible causes of the event are known and modeled by the parents of the Noisy-Or node may not always hold. There are domains, such as medicine, where the causes of events are not entirely known. It is, therefore, desirable to have a mechanism to make such incompleteness explicit. *Leak* nodes provide a way to do exactly that.

A *leak* node is simply an extra node that is added to the parent set of a Noisy-Or or a Noisy-And node. Figure 2.6 shows an example of a leak node used with a Noisy-Or node. Leak nodes are assigned prior probabilities that reflect the degree of incompleteness of the model. The probability associated with the link from a leak node to a Noisy-Or/Noisy-And node reflects the degree to which the latter is affected by unknown influences. Once they have been introduced in a network, leak nodes are treated like any other node for purposes of reasoning and inference.

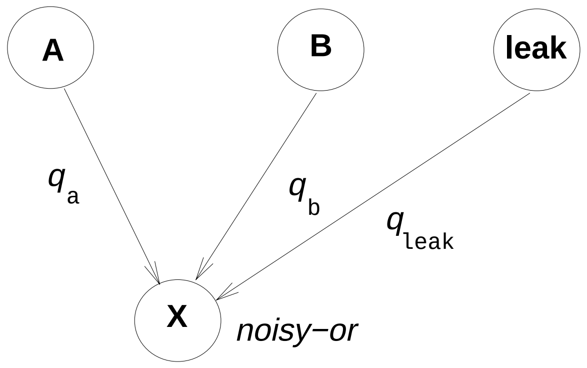


Figure 2.6: Example of a Leak node

2.2 Multi-layered Feed-Forward Neural Networks

Our technique for revising the parameters of a Bayesian network uses a gradient descent, back-propagation algorithm that is based on the technique used to train multi-layer feedforward neural networks (McClelland & Rumelhart, 1988). Such networks represent functions that map a set of input values to a set of output values, and can be trained inductively to learn specific functions. Here, we provide an overview of the backpropagation algorithm used to train such neural networks. In addition to providing the background for the parameter revision algorithm to be discussed later on, this will also help in a better understanding of our experimental evaluation, where we compare our technique to several other inductive learning techniques including backpropagation with neural networks.

Figure 2.7 shows an example of a multi-layered feed-forward neural network (Hertz, Krogh, & Palmer, 1991). The output units are denoted by O_i and the input units by I_k. The layer between the input and the output layers is the hidden layer, whose units are denoted by H_j. The connections between the units in the input and the hidden layers are denoted by w_{jk} and those between the hidden and output layers are denoted by W_{ij}.

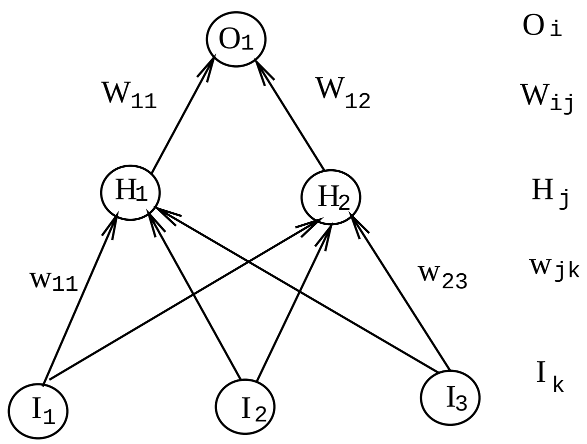


Figure 2.7: Multi-layered Feed-Forward Neural Network - Example

When the input units of a network are clamped to particular values, the output of the network is computed by propagating these values through the hidden layers to the output layer. The activation of each unit is a function of a weighted combination of the values of all the units feeding into it. Given a pattern μ, each unit computes a function of the weighted sum of all its inputs, as shown below.

$$h_j^\mu = \sum w_{jk} I_k \tag{2.6}$$

The activation or the output of each hidden unit is given by

$$V_j^\mu = g(h_j^\mu) \tag{2.7}$$

where $g(h)$ is a thresholding function, or a continuous sigmoid function. The activations of the output units are computed similarly.

The backpropagation algorithm is often used to train networks, given some data mapping a set of values for the input variables with a set of values for the output variables. This algorithm uses a gradient descent approach to modify the weights in the network so as to minimize the *mean squared error* of the network with respect to the data, defined as:

$$E[\mathbf{w}] = 1/2 \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2 \tag{2.8}$$

where ζ_i^μ is the desired value of the i-th output variable for pattern μ of the data and O_i^μ is the actual output of the i-th output unit for the same pattern. The backpropagation algorithm then changes each weight w_{ij} by an amount Δw_{ij} proportional to the gradient of E given by:

$$\Delta w_{ij} = -\eta \partial E / \partial w_{ij} \tag{2.9}$$

where η is a parameter that controls the *learning rate.* Deriving an accurate gradient of the error function is, therefore, a crucial aspect of the backpropagation approach.

At the start of training, the network is initialized with some (typically random) weights.[1] For each pattern in the data, the inputs of the network are clamped with the input values specified in the pattern. These values are propagated forward to the output units. The error between the output values predicted by the network and those specified in the data is propagated back through the hidden layers to the input layer. The error propagated to each unit is used to modify the weights of the connections feeding into the unit, according to the learning rule given by Equation 2.9. This process is repeatedly iterated over all the patterns in the data until a desired level of convergence is reached.

Knowing when to stop is often a critical issue while training neural networks. Too many iterations of training often results in overfitting, so that the network fits outliers and noisy data points that do not reflect the true distribution being learned. This hurts the network's ability to generalize to unseen cases (Holder, 1991). Training for too few iterations, on the other hand, results in poor convergence on the training data, which also hurts the network's ability to generalize. Several techniques have been proposed to address this issue (Weigand, Huberman, & Rumelhart, 1990; Holder, 1991). One common approach is to set aside a portion of the training data to serve as a *validation set* used to track the generalization performance of the network, train the network on the reduced training data, and stop training whenever the generalization performance on the validation set shows no improvement for a certain number of cycles. The network is then re-trained on the full training set for the same number of cycles as on the reduced training set. Another approach is to perform k-fold internal cross-validation, where the training data is partitioned into k equal subsets. k networks are trained, each using $k - 1$ of the subsets for training and the remaining one for testing. The generalization performance of each network on the internal test sets is monitored at the end of each training cycle, and the average generalization error is computed for each cycle. The number of training epochs that shows the least average generalization error is picked as the number of epochs needed to train on the full training set.

2.3 Theory Refinement

Early experience with expert systems and rule-based systems quickly revealed a *knowledge-acquisition bottleneck*, i.e. the fact that extracting knowledge from experts to be encoded into these systems can be very time-consuming, and that such knowledge is often incomplete or not entirely accurate. One way of addressing this problem is via *inductive learning* (Quinlan, 1993; Langley & Simon, 1995; Mitchell, 1997), where the idea is to minimize the amount of knowledge to be extracted from an expert by having the system extract knowledge autonomously from examples. For instance, in order to encode classification knowledge, the system would be presented with instances of classes, and it would *learn* to generalize from these examples to new classification problems. While many of the techniques developed for this purpose have been fairly successful (Langley & Simon, 1995; Mitchell, 1997), they typically require large amounts of data to learn accurate classifiers. Therefore, several techniques have been developed that take a more collaborative approach to knowledge acquisition. Commonly known as *theory revision* or *theory refinement*, such an approach assumes that the learner can be provided with some initial, approximate knowledge, usually acquired from an expert, and all it has to do is refine this knowledge, using the given data, to make it more

[1]Sometimes the initial weights are determined using domain knowledge.

accurate.

Given an initial domain theory which may not be complete or correct, theory refinement systems modify the initial theory (minimally) to improve its accuracy on a given data set. Techniques have been developed to revise and improve propositional and first-order theories (Ourston & Mooney, 1994; Cohen, 1992; Wogulis & Pazzani, 1993; Wogulis, 1994; Baffes & Mooney, 1993; Baffes, 1994; Richards & Mooney, 1995), as well as to revise hybrid rule-bases that combine logical and numerical representations (Towell & Shavlik, 1994; Opitz & Shavlik, 1993; Opitz, 1995a; Mahoney & Mooney, 1993; Mahoney, 1996). Experiments show that theory refinement systems can learn accurate theories with less data than purely inductive systems. The idea that similar techniques can be used to effectively learn Bayesian networks is the cornerstone of this research. Here we briefly describe some techniques for revising logical rule-bases, as well as some hybrid techniques that combine logical and numeric representations to learn theories that are more accurate than logical theories. All of these systems are focused on learning for classification, and assume that the data is described in terms of a set of *input features*, and that the goal is to learn to classify data into a set of *output classes.*

2.3.1 Logic-Based Approaches

The goal of theory refinement is to modify an existing imperfect domain theory to make it consistent with a set of data. For logical theories, this can be more precisely defined as follows:

- **Given:** An initial theory T and a set of positive examples P and a set negative examples N, where P and N are restricted to ground formulae.
- **Find:** A "minimally revised" consistent theory T' such that $\forall p \in P : T' \vdash p$ and $\forall n \in N : T' \not\vdash n$.

Generally, examples are ground Horn-clauses of the form $C \leftarrow B_1, \ldots, B_n$, where the body, B, gives a description of a case and the head, C, gives a conclusion or classification that should logically follow from this description (or should not follow in the case of a negative example). Revising a logical theory may require both adding and removing clauses as well as adding or removing literals from existing clauses. Generally, the ideal goal is to make the minimal syntactic change to the existing theory (Wogulis & Pazzani, 1993; Mooney, 1995). Unfortunately, this task is computationally intractable; therefore, in practice, heuristic search methods must be used to approximate minimal syntactic change.

Several theory refinement systems use abduction on individual examples to locate faults in a theory and suggest repairs (Ourston & Mooney, 1990; Ourston, 1991; Ourston & Mooney, 1994; Wogulis & Pazzani, 1993; Wogulis, 1994; Baffes & Mooney, 1993; Baffes, 1994; Baffes & Mooney, 1996; Brunk, 1996). Abduction is the process of inferring cause from effect or constructing a specific set of assumptions that explain observed events. Each of the above systems uses abduction in a slightly different way, but the following discussion summarizes the basic approach. For each individual positive example that is not derivable from the current theory, abduction is applied to determine a set of assumptions that would allow it to be proved. These assumptions can then be used to make suggestions for modifying the theory. One potential repair is to learn a new rule for the assumed proposition so that it could be inferred from other known facts about the example.

Another potential repair is to remove the assumed proposition from the list of antecedents of the rule in which it appears in the abductive explanation of the example. For example, consider the theory:

```
P(X) ← R(X), Q(X).
Q(X) ← S(X), T(X).
```

and the unprovable positive example:

```
P(a) ← R(a), S(a), V(a).
```

Abduction would find that the assumption `T(a)` makes this positive example provable. Therefore, two possible refinements to the theory are to remove the literal `T(X)` from the second clause in the theory, or to learn a new clause for `T(X)`, such as

```
T(X) ← V(X).
```

Another possible abductive assumption is `Q(a)`, suggesting the possible revisions of removing `Q(X)` from the first clause or learning a new clause for `Q(X)` such as

```
Q(X) ← V(X).
```

or

```
Q(X) ← S(X), V(X).
```

In order to find a small set of repairs that allow *all* of the positive examples to be proved, a greedy set covering algorithm can be used to select a small subset of the union of repair points suggested by the abductive explanations of individual positive examples, such that the resulting subset covers all of the positive examples. If simply deleting literals from a clause causes negative examples to be covered, inductive methods can be used to learn a new clause that is consistent with the negative examples. Continuing the example, assume the positive examples are:

```
P(a) ← R(a), S(a), V(a), W(a).
P(b) ← R(b), V(b), W(b).
```

and the negative examples are:

```
P(c) ← R(c), S(c).
P(d) ← R(d), W(d).
```

The abductive assumptions `Q(a)` and `Q(b)` are generated for the first and second positive examples respectively. Therefore, making a repair to the `Q` predicate would cover both cases. Note that the previously mentioned potential repairs to `T` would not cover the second example since the abductive assumption `T(b)` is not sufficient (both `T(b)` and `S(b)` must be assumed). Since a repair to the single predicate `Q` covers both positive examples, it is chosen. However, deleting the antecedent `Q(x)` from the first clause of the original theory would allow both of the negative examples to be proven.

Therefore, a new clause for `Q` is needed. Positive examples for `Q` are the required abductive assumptions `Q(a)` and `Q(b)`. Negative examples are `Q(c)` and `Q(d)` since these assumptions would allow the negative examples to be derived. Given the descriptions provided for `a, b, c` and `d` in the examples, typical inductive techniques for learning Horn-clause rules would induce the new clause:

```
Q(X) ← V(X).
```

since this is the simplest clause that covers both of the positive examples without covering either of the negatives. Note that although the alternative, equally-simple clause

```
Q(X) ← W(X)
```

covers both positive examples, it also covers the negative example `Q(d)`.

The EITHER (Ourston & Mooney, 1990, 1994; Ourston, 1991) and NEITHER (Baffes & Mooney, 1993; Baffes, 1994) theory refinement systems allow multiple assumptions in order to prove an example, preferring more specific assumptions, i.e. they employ *most-specific abduction* (Cox & Pietrzykowski, 1987). AUDREY (Wogulis, 1991), AUDREY II (Wogulis & Pazzani, 1993), A3 (Wogulis, 1994), and CLARUS (Brunk, 1996) are a series of theory refinement systems that make a "single-fault assumption" during abduction. For each positive example, they find a single most-specific assumption that makes the example provable. Different constraints on abduction may result in different repairs being chosen, effecting the level of specificity at which the theory is refined. EITHER and NEITHER prefer making changes to the more specific aspects of the theory rather than modifying the top-level rules.

This general approach of using abduction to suggest theory repairs has proven quite successful at revising several real-world knowledge bases. The systems referenced above have significantly improved the accuracy of knowledge bases for detecting special DNA sequences called promoters that signal the start of a new gene (Ourston & Mooney, 1994; Baffes & Mooney, 1993), diagnosing diseased soybean plants (Ourston & Mooney, 1994), and determining when repayment is due on a student loan (Brunk, 1996). The approach has also been successfully employed to construct rule-based models of student knowledge for over 50 students using an intelligent tutoring system for teaching concepts in C++ programming (Baffes, 1994; Baffes & Mooney, 1996). In this application, theory refinement was used to modify correct knowledge of the domain to account for errors individual students made on a set of sample test questions. The resulting modifications to the correct knowledge base were then used to generate tailored instructional feedback for each student. In all of these cases, experiments with real training and test data were used to demonstrate that theory refinement resulted in improved performance on novel, independent test data and generated more accurate knowledge than raw induction from the data alone. These results clearly demonstrate the utility of integrating abduction and induction for theory refinement.

2.3.2 Hybrid Approaches

Hybrid approaches combine logical and numerical representations in order to learn accurate classifiers. While the initial theory is still specified using propositional logic, these techniques convert the theory into a numerical representation such as neural networks, and revise the new form of the theory. Most of these algorithms follow the high-level strategy of first revising the numerical component of the representation (called *parameters* in this discussion) using an iterative algorithm, and then revising the logical component (called *structure*) if needed, and repeating this process until a desired level of convergence is achieved (Figure 2.8). Experiments show that these approaches produce more accurate theories than the purely logic-based approaches on several real-world problems.

Given: An initial approximate theory in the form of propositional Horn-clause rules, and a set of training data.
Output: A hybrid representation that accurately classifies the data.
Algorithm:

```
1. Convert the theory into the hybrid representation.
2. Repeat until accuracy on training data ceases to improve:
    (a) Revise the parameters of the representation.
    (b) If not 100% training accuracy, revise the structure of the
        representation.
```

Figure 2.8: High-level strategy used by hybrid approaches

KBANN (Towell & Shavlik, 1994) provides a way of combining symbolic and connectionist representations. It uses an initial propositional rule-base to determine the structure of a multi-layer feedforward neural network with units that simulate the conjunctions and disjunctions in the rule-base. In addition, all the remaining features in the data not included in the initial domain theory are added to the input layer. Each unit in layer n is connected to each unit in layer $n+1$, with links not justified by the domain theory initialized to low weights, and the network is trained on a given data set using the backpropagation algorithm described earlier (Section 2.2). This approach has been shown to be effective on a large number of real-world problems such as *DNA promoter recognition* (Noordewier et al., 1991) and *DNA splice-junction recognition* (Noordewier et al., 1991). Techniques have also been developed to extract a rule-base from the trained network, in order to make the revised knowledge more comprehensible (Towell & Shavlik, 1993; Craven & Shavlik, 1996; Craven, 1996).

KBANN relies on weight adjustment to revise the initial theory. This approach was found to be insufficient for some problems. Although KBANN could effectively revise rules with extra antecedents (by deleting some links in the network), it was found to be not as effective in revising domain theories with missing rules (Opitz, 1995a). TOPGEN was developed as a technique to add new nodes to a network constructed by KBANN (Opitz, 1995a). Following the strategy shown in Figure 2.8, it first uses KBANN to revise an initial theory. If the trained network still misclassifies some examples, TOPGEN repairs the theory by adding a node to the network and passes on the new network to KBANN for further training. This is repeated until further training does not improve the generalization of the network. TOPGEN adds nodes by approximating the trained KBANN network by a symbolic theory, and propagating the blame for misclassified examples to the internal nodes. Depending on the type of the node with the highest blame (And/Or) and the nature of the blame (overly general/overly specific), a new node is introduced as a child/sibling of the node, which is linked to all input features. TOPGEN has been shown to learn networks that generalize better than KBANN for several real-world problems including *DNA promoter recognition*, and *DNA splice-junction recognition*.

Many expert systems use *certainty factors* to represent uncertainty (Buchanan & Shortliffe, 1984). *Certainty factors* are numbers associated with the rules in a rule-base that informally represent the degree to which each antecedent of a rule confirms the consequent. These certainty factors can be combined to compute the degree of certainty associated with any conclusion derived from the rule-base. RAPTURE (Mahoney & Mooney, 1993; Mahoney, 1996) is a system for revising rule-bases with associated certainty factors. Given an initial rule-base, RAPTURE maps the rules into a neural network such that the weights on the links are the certainty factors associated with the rule-base. RAPTURE can revise the initial domain theory in three ways to improve its predictive accuracy. It uses the approach shown in Figure 2.8 and first revises the certainty factors associated with the network, using the gradient descent backpropagation algorithm described earlier (McClelland & Rumelhart, 1988). If this does not improve the accuracy of the network sufficiently, it finds input features to link to each of the output classes being mis-labeled. These two steps are repeated until further revision fails to improve the training accuracy of the network. If the network still misclassifies some examples, RAPTURE uses the UPSTART algorithm (Frean, 1990) to add new hidden units to the network. For each output class, two new units are trained to differentiate between misclassified examples that are specified to be positive examples of the class, and the misclassified examples that are specified as negative examples of the class. These new units are then linked to the corresponding output units. This approach to theory refinement has been successfully applied to problems such as *DNA promoter recognition*, *soybean disease classification* (Michalski & Chilausky, 1980), *DNA splice-junction recognition*, and the MYCIN expert system rule-base (Buchanan & Shortliffe, 1984).

2.4 Naive Bayes

A *Naive Bayes* classifier is a Bayesian network with two layers of nodes.[2] The root node represents the class variable and is fully connected to a second layer of nodes that represent the attributes or features associated with the classification problem. The attributes are assumed to be independent of each other, given the value of the class variable. This architecture, combined with the independence assumption, leads to a very fast and efficient algorithm for learning the weights on the links of the network. Many studies have shown that this algorithm is very effective in learning accurate classifiers. In fact, this algorithm produces the best classifiers for many of the standard machine learning benchmark problems (Kohavi, Becker, & Sommerfield, 1997; Domingos & Pazzani, 1997).

Consider the problem of determining whether or not an object is a cup, given the observation that it is small and concave, that it is red in color, and that it has a handle. The Naive Bayes classifier (Figure 2.9), given these observations, classifies the object according to the probability $P(Cup = true \mid Size = small, Shape = concave, Color = red, Has\_handle = true)$. For brevity, we will drop the attribute names in the ensuing discussion, and use the standard representation for Boolean attributes. By Bayes rule,

$$\begin{aligned} &P(cup \mid small, concave, red, has\_handle) = \\ &\quad \frac{P(cup) \times P(small, \ldots, has\_handle \mid cup)}{P(small, \ldots, has\_handle)} \end{aligned} \tag{2.10}$$

Since,

$$P(cup \mid small, \ldots, has\_handle) + P(\neg cup \mid small, \ldots, has\_handle) = 1 \tag{2.11}$$

[2]Although the Naive Bayes classifier is a Bayesian network, we treat it here in a separate section rather than include it in the next section, because we want to describe it in greater detail than rest of the Bayesian network learning algorithms.

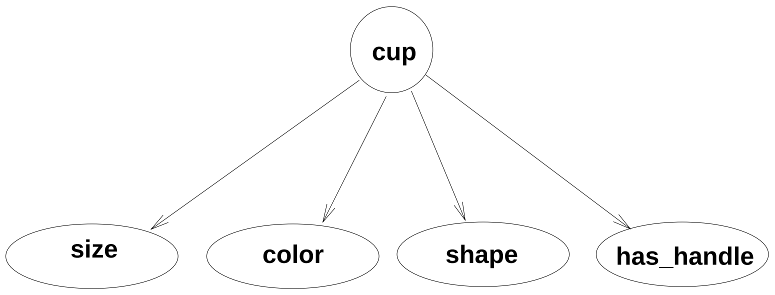


Figure 2.9: Example of Naive Bayes classifier

$$\begin{aligned} P(small, \ldots, has\_handle) &= P(cup) \times P(small, \ldots, has\_handle \mid cup) + \\ &\quad P(\neg cup) \times P(small, \ldots, has\_handle \mid \neg cup) \end{aligned} \tag{2.12}$$

and maybe treated as a normalization factor. Assuming that the observed features of the object, i.e. color, shape etc., are independent, given that the object is a cup, Equation 2.10 can be re-written as:

$$\begin{aligned} P(cup \mid small, \ldots, has\_handle) &= P(cup) \times P(small \mid cup) \times P(concave \mid cup) \\ &\quad \times P(red \mid cup) \times P(has\_handle \mid cup) \end{aligned} \tag{2.13}$$

In order to classify the object, it is sufficient to specify the probabilities of each of the observed features given that the object is a cup, the probabilities of each of the observed features given that the object is not a cup, and the prior probability that the object is a cup.

In general, given an example with n attributes to be classified into one of k classes, the goal of a Naive Bayes classifier is to determine the class C_i with the highest conditional probability $P(C_i \mid \wedge a_j)$, where $\wedge a_j$ stands for the conjunction of the attributes $a_1 \cdots a_n$. Applying Bayes law, and assuming that the attributes are independent, given the value of the class variable, this probability is proportional to:

$$P(\wedge a_j \mid C_i) = \prod_{j=1}^{n} P(a_j|C_i) \times P(C_i) \tag{2.14}$$

The problem of learning a Naive Bayes classifier from examples is simply that of estimating the conditional probabilities, $P(a_j \mid C_i)$, for each combination of attribute a_j and class C_i, and the prior probabilities of each class. The former can be estimated just by counting the number times these features take on specific values for different values of the class variable, leading to a learning algorithm that is polynomial in the number of examples. This approach, however, leads to undesirable outcomes when an attribute value a_j does not occur together with a given class value C_i, resulting in a zero estimate for $P(a_j \mid C_i)$. In such a case, class C_i would be summarily rejected whenever a_j is true, leading to the undesirable situation of the outcome being determined by a single attribute. One commonly used technique for avoiding this problem is to replace zero estimates by very small positive values, usually by a factor that is inversely proportional to the number of training examples. Clark and Niblett (1989) and Domingos and Pazzani (1996) replace a zero estimate for $P(a_j \mid C_i)$ by $\frac{P(C_i)}{m}$, where m is the number of training examples. Recently, Kohavi et al. (1997) have proposed a technique for smoothing out the estimate of the conditional probabilities by setting $P(a_j \mid C_i)$ to $\frac{(N+f)}{(n+kf)}$, where N is the number of examples of class C_i for which attribute a_j is true, n is the number of examples of class C_i, f is a pre-defined constant, and k is the number of classes in the problem. Experiments show that the *Laplace* estimate, as this is called, leads to better results compared to other techniques for handling the problem of zero estimates (Kohavi et al., 1997).

Experiments on several benchmark learning problems show that the algorithm for learning Naive Bayes classifiers, henceforth referred to as the *Naive Bayes* algorithm, is very effective on small data-sets with a few hundred examples or less, producing accurate classifiers that are comparable, and sometimes better than the ones learned using other induction algorithms such as C4.5 (Quinlan, 1993). However, it does not perform as well on large data sets with thousands of examples. This behaviour can be explained in terms of the bias-variance decomposition of error (Kohavi & Wolpert, 1996; Geman, Bienenstock, & Doursat, 1992), where the error of a learned classifier is viewed as the sum of two components: the bias and the variance. Bias measures the degree to which the learning algorithm can fit the data, with lower values indicating higher bias, and variance measures the stability of the learning algorithm, i.e. the degree to which the results of the learning algorithm vary according to the training data, with lower values indicating lower variance. The Naive Bayes algorithm has strong bias because the space of hypothesis it considers is very limited and low variance as small changes to the training data rarely cause large changes in the results (Kohavi et al., 1997). Variance is a more critical issue for smaller training sets, whereas bias is a more critical issue for larger data sets. Thus, the low variance of the Naive Bayes learning algorithm explains its effectiveness on smaller data sets, and its high bias explains why it is not as effective on larger training sets.

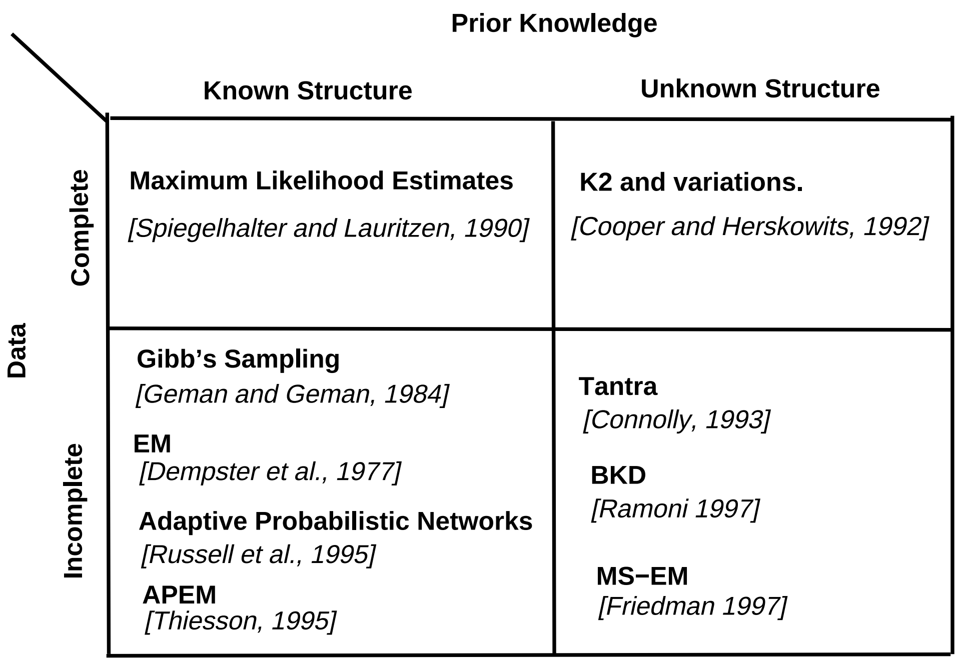
2.5 Learning Bayesian Networks

Recent years has witnessed a surge of interest in the problem of learning Bayesian networks. A common idea underlying most of the existing approaches is that of learning a network that maximizes $P(B \mid D)$, i.e. the likelihood of the network given the data. By Bayes rule:

$$P(B \mid D) = P(D \mid B) \times P(B) \tag{2.15}$$

Assuming that all networks are equally likely a-priori makes $P(B)$ a constant with respect to all networks, and the problem of learning a network that maximizes $P(B \mid D)$ becomes equivalent to that of learning a network that maximizes $P(D \mid B)$. Most techniques for learning Bayesian networks introduce a scoring metric, based on the goal of maximizing $P(D \mid B)$, for evaluating each network with respect to the data, and search for the best network according to this metric. Heckerman (1995) provides a detailed discussion of the principles underlying several techniques for learning Bayesian networks. In this section, we will briefly describe some representative techniques.

As discussed in Chapter 1, according to the assumptions that algorithms for learning Bayesian networks make about prior knowledge of the structure of the network, and the completeness of data, they can be classified as follows: techniques that assume that they are given a *correct structure* and *complete data*, techniques that assume that they are given the *correct structure*



Data \ Prior Knowledge	Known Structure	Unknown Structure
Complete	**Maximum Likelihood Estimates** *[Spiegelhalter and Lauritzen, 1990]*	**K2 and variations.** *[Cooper and Herskowits, 1992]*
Incomplete	**Gibb's Sampling** *[Geman and Geman, 1984]* **EM** *[Dempster et al., 1977]* **Adaptive Probabilistic Networks** *[Russell et al., 1995]* **APEM** *[Thiesson, 1995]*	**Tantra** *[Connolly, 1993]* **BKD** *[Ramoni 1997]* **MS−EM** *[Friedman 1997]*

Figure 2.10: Techniques for learning Bayesian networks

and possibly *incomplete data*, techniques that assume that the *structure is unknown*, but the data is *complete*, and techniques that assume that the *structure is unknown* and the data is possibly *incomplete*. Figure 2.10 shows some of the existing Bayesian network learning techniques classified according to this criteria. We will discuss some of these techniques here, while the rest will be discussed in the chapter on related work (Chapter 7).

When the structure of a network is known, and there are no missing or hidden variables, the goal of maximizing $P(D \mid B)$ can be achieved simply by using the data to calculate the frequencies of various values of the variables given various values of their parents in the network (Spiegelhalter & Lauritzen, 1990). These frequencies estimate the conditional probability tables associated with each node. The Naive Bayes learning algorithm described in the previous section is, in fact, an example of such an approach.

There are several techniques for estimating the parameters of a network of a known structure in the presence of hidden variables and missing data. EM (Dempster et al., 1977; Lauritzen, 1995), or *Expectation Minimization*, is an iterative technique which starts out with some initial values for the parameters that may be random or based on some prior knowledge. The data set is then augmented with the most likely values for the missing data, estimated from the parameters from the current iteration, and used to estimate the parameters for the next iteration. This process is repeated until convergence.

Russell et al. (1995) proposed APN as a technique for learning the parameters of a Bayesian network with hidden variables. It uses gradient descent to optimize $\ln P_w(D|B)$, i.e. the log of the probability assigned by the network to the data when the parameters are set to w. The algorithm iterates over each example in the data set several times until convergence. For each example, it places the observed values as evidence, and propagates the evidence throughout the network. The parameters of the network are then updated using the following equation to compute the gradients:

$$\frac{\partial \ln P_w(D)}{\partial w_{ijk}} = \sum_{l=1}^{m} \frac{P_w(x_{ij}, u_{ik} \mid D_l)}{w_{ijk}} \tag{2.16}$$

where m is the number of training instances, x_{ij} is the jth possible assignment to variable X_i, u_{ik} is the kth possible value assignment to the parents of X_i, w_{ijk} is the probability that variable X_i takes on its jth possible assignment given that its parents U_i take on their kth possible assignment. $P_w(x_{ij}, u_{ik} \mid D_l)$ is computed as a by-product of inferring the probabilities of the variables in the network, given the evidence in D_l. Note that there is no need to back-propagate the gradients. All the information for computing the gradient of the weight on a link is present in the nodes at the head of the link and its parents.

K2 (Cooper & Herskovits, 1992) was one of the first techniques developed to learn both the structure and the parameter of a network. It uses the *belief scoring* function for evaluating Bayesian networks with respect to the data as a basis for a greedy, hill-climbing search to learn a network that closely models the probability distribution in the data. Assuming that all the variables in the data set are discrete, that there are no variables with missing values, that all structures are equally likely a-priori, and that the examples in the data set are independent of each other, the *belief scoring* function is defined as:

$$Score(B, D) = c \times \prod_{i=1}^{n} \prod_{j} = 1^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod N_{ijk}! \tag{2.17}$$

where c is a constant, r_i is the number of values associated with variable X_i, q_i is the number of possible instantiations of the parents of the variable X_i, N_{ijk} is the number of cases in D where X_i takes on its k-th value and its parents take on their j-th instantiation, and N_{ij} is $\sum_{k=1}^{r_i} N_{ijk}$. To maximize $Score(B, D)$, it is sufficient to find a parent set for each variable that maximizes the inner product.

$$max_B[Score(B, D)] = c \times \prod_{i=1}^{n} max_{\pi_i} \left(\prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \right) \tag{2.18}$$

K2 starts with a network where there is one node per variable in the data, and none of the nodes has any parents. K2 then incrementally adds a link that maximizes the inner product above. It stops when further addition of a link fails to increase $Score(B, D)$. Several variations and extensions to K2 have since been proposed (Buntine, 1991; Heckerman et al., 1994; Provan & Singh, 1994).

Learning both the structure and the parameters of a Bayesian network in the presence of incomplete data is currently a very active area of research. The most recent and general-purpose approach, MS-EM (Friedman, 1997) combines hill-climbing with an EM-like approach for estimating missing and hidden values. It uses a scoring metric based on the principle of *minimum description length (MDL)* (Rissanen, 1978; Lam & Bacchus, 1994a) as the criteria for searching for networks that best match the data. This metric is defined as:

$$Score(B, D) = L(B, D) - \frac{logN}{2} \#(B) \tag{2.19}$$

where $L(B,D)$ is the log-likelihood of B given D, and $\#(B)$ is the number of parameters in the network. The log-likelihood of B given D is defined as

$$L(B,D) = \sum_{i=1}^{N} log(d^i)$$

where $D = \{d_1 \cdots d_n\}$. The minimum description length principle (Rissanen, 1978) provides a way to balance the trade-off between learning to fit the data accurately, and overfitting the data by learning too complex a network. Like the *belief scoring* metric (Equation 2.17) the above score can be decomposed into computations that are local to each node and its parents when the data is complete. However, this is not possible with incomplete data. Therefore, like EM, MS-EM attempts to maximize the *expected* score by taking expectations over all the possible values that the missing values may take. Friedman (1997) show that this expected score is decomposable in a manner similar to Equation 2.17. The MS-EM algorithm starts out with an initial network that may be random or based on some prior knowledge. It first uses EM for a specified number of steps to estimate the parameters of the network. It then modifies the structure of the network by evaluating alternate modifications to pick the one with the best expected score. The parameter revision and structure modification steps are repeated until convergence.

Both K2 and MS-EM are iterative algorithms that add one link at each iteration to learn the structure of a network. Selecting a link to be added involves selecting both a node in the network to be expanded by adding a new parent, and a node to be added as a new parent. At each iteration, they consider all possible ways of adding a link to a network, evaluate each addition using a scoring metric, and add the link that best improves the score. Thus, they are based on the *generate then test* paradigm where all the possible successors to the current structure are generated, and the data is only used to select among them (Mitchell, 1997). No effort is made to use the information in the data to restrict the set of possible successors to be evaluated. Since each evaluation requires considering the entire data set, restricting the number of networks to be evaluated would reduce the computational effort required by the learning algorithm. This would be especially beneficial when the data includes missing values and hidden variables, since the evaluation function in such a case involves marginalizing over these missing values and is hence computationally expensive (Friedman, 1997). Extending these approaches to discover hidden variables to the network would further increase the search space, and would benefit greatly from strategies to restrict the number of candidate networks to be evaluated by the learning algorithm.

Most of the focus of research in this field has been on techniques for learning networks that model the distribution underlying the data, without regard to the specific intended task of the network. Often Bayesian networks are built for a specific task such as classification. Only recently has there been any effort to study the advantages of learning networks that are optimized for the specific tasks for which they are intended. Friedman and Goldszmidt (1996) show that, when the goal is to learn a Bayesian network to be used as a classifier, it is better to use techniques that optimize the probability distribution of the class variables conditioned on the attributes. Greiner, Grove, and Schuurmans (1997) also argue in favor of learning task-specific networks, and propose a technique for learning the parameters of a network that seeks to optimize the network for a given task.

A standard practice in the field of learning Bayesian networks is to evaluate a technique by comparing the learned network with the original network, called the *gold standard* (Heckerman, 1995). This assumes that the target network is known ahead of time. Given the target network, the practice is to generate data from the network, learn a Bayesian network from the generated data, and compare the learned network with the original. One of the data sets that is used as a standard for such experiments is the ALARM data set, which is a database of cases generated from a network designed to assist in monitoring the heart rate of a patient (Beinlich, Suermondt, Chavez, & Cooper, 1989; Cooper & Herskovits, 1992). Techniques such as APN (Russell et al., 1995), and MS-EM (Friedman, 1997) have been evaluated on data generated from a network for car insurance risk estimation (Musick, 1994). While evaluating learning algorithms on artificially generated data sets demonstrates the ability of a learning algorithm to re-construct a Bayesian network, it does not demonstrate its ability to learn in real-world learning situations, and ignores altogether the issue of the appropriateness of the representation for modeling real-world problems. However, Bayesian network learning algorithms are increasingly being evaluated on real data sets (Provan & Singh, 1994; Friedman & Goldszmidt, 1996) and with the growing number of applications of Bayesian networks real-world problems (Burnell & Horovitz, 1995; Fung & Del Favero, 1995), it is hoped that the trend towards evaluating learning techniques on real data sets will continue to grow as well.

Many of the above approaches provide a way of specifying prior knowledge about the structure of the network in the form of prior probabilities of network structures ($P(B)$ in Equation 2.15). Prior knowledge can be directly introduced into the MS-EM algorithm in the form of an initial network. Research in the field of machine learning has demonstrated that using an approximate domain theory as a starting point for learning yields better results than learning from scratch (Ourston & Mooney, 1990; Towell et al., 1990; Thompson et al., 1991). However, there has been no effort to study the effectiveness of using the above techniques for theory refinement. Most of the research in learning Bayesian networks is focused on learning from scratch, and barring some exceptions (Buntine, 1991; Lam & Bacchus, 1994b), the issue of theory refinement has largely been ignored.

Chapter 3

BANNER: An Overview

In this chapter, we introduce BANNER, which is a technique for revising Bayesian networks with Noisy-Or and Noisy-And nodes. Given an initial theory and some data, BANNER produces a Bayesian network with improved classification accuracy. The initial theory could either be a Bayesian network with Noisy-Or and Noisy-And nodes, or a set of propositional rules. In the latter case, the propositional rules are converted into a Bayesian network with Noisy-Or and Noisy-And nodes. BANNER is geared to learn networks for classification, where, given the value of a set of observations, called input features, the network is used to predict whether or not it belongs to a certain class. The training data is assumed to consist of a set of labeled examples, each specifying a set of values for the input features and its classification. Classification accuracy is defined as the number of examples that are correctly classified. An example is considered to belong to a certain class, when the probability of the node corresponding to the class, given the input features, is greater than 0.5.

The following section will present an overview of the algorithm used by BANNER. This will be followed by a discussion of the applicability and the limitations of BANNER. Section 3.3 will discuss the procedure for converting a logical propositional Horn-clause rule-base into a Bayesian network with Noisy-Or/And nodes.

3.1 Overview of the Algorithm

Like RAPTURE (Mahoney & Mooney, 1993, 1994), BANNER uses a two-tiered approach to the problem of learning a Bayesian network from partial specification. Given some data, BANNER first tries to improve the network by revising the parameters of the network. If the network still does not fit the training data, the structure of the network is modified to find the network with the highest predictive accuracy. These steps are repeated until further revision does not lead to an improvement in accuracy. The structure of a Bayesian network represents correlations or dependencies between the variables in the domain, and the parameters represent the degree of dependency between variables. Thus, modifying the structure of network is a more fundamental change to the model of the domain than modifying just the parameters. Since one of the goals of theory refinement is to modify the initial theory minimally, BANNER first tries to fit the data with parameter revision before considering structure revision. The two main components of BANNER are:

1. The parameter revision component, which assumes that the structure of the network is correct, and is concerned with modifying the parameters of the network to improve predictive accuracy. BANNER is designed so that this component is quite separate from the structure revision component. Thus, it is possible to plug-in one's favorite parameter revision algorithm without having to change the rest of the system. For our experiments, we have used two different parameter revision techniques: BANNER-PR, and a variant of APN calledC-APN. BANNER-PR (Ramachandran & Mooney, 1996b) uses a gradient descent backpropagation algorithm similar to that used in multi-layered feedforward networks. It has been designed specifically to take advantage of the computational efficiency offered by Noisy-Or and Noisy-And nodes. However, as we discussed later in the chapter, it is restricted in its applicability. C-APN, based on APN (Russell et al., 1995) (Section 2.5), is a more general technique for revising the parameters of a Bayesian network which also uses gradient descent. Whereas APN learns to optimize the likelihood of the entire data being generated by the network, C-APN learns a network that is optimized to estimate the probability of certain class variables given some evidence. It does not, however, exploit the linearity of specification resulting from the use of Noisy-Or and Noisy-And nodes and hence is much slower to converge. We will elaborate these techniques further in Chapter 4.

2. The second component, BANNER-SR is concerned with minimally modifying the structure of the network in order to improve its classification accuracy. Much like logical theory refinement techniques such as EITHER (Ourston & Mooney, 1994), the algorithm for this component uses abductive inference to attribute failures in classification to specific portions of the network, and to find appropriate revisions to correct these failures. It is based on the idea of augmenting the Noisy-Or/And nodes in the network with leak nodes (Section 2.1.2) to serve as indicators of possible faults in the network. Once the network is so augmented, the algorithm performs abductive inference on each misclassified example to select a subset of leak nodes that explain the error. Such information, gathered for each misclassified example, is used to select a minimal set of nodes and links in the network to be modified. We will discuss this algorithm further in Chapter 5.

Figure 3.1 shows the overall architecture of our system. As mentioned earlier, the modularity of the design, and the separation between the components allow us to replace the algorithm implementing each component without affecting the entire system.

3.2 Limitations of BANNER, BANNER-PR and C-APN

Bayesian networks offer various choices in terms of the kinds of local distributions that can be modeled, the architecture of the networks, and the kind of reasoning that can be performed. Depending on the problem, one may wish to model it as a Bayesian network with unrestricted interactions of influences, or as a network with Noisy-Or and Noisy-And nodes (or other special restricted models). Although, the former is more general, restrictive models like the Noisy-Or model are considerably more efficient and require fewer parameters. Similarly, a problem may require networks with loops. In such a case, inference would be computationally more expensive than for problems that can be modeled by networks without loops. Finally, one may wish to build a network to be used for

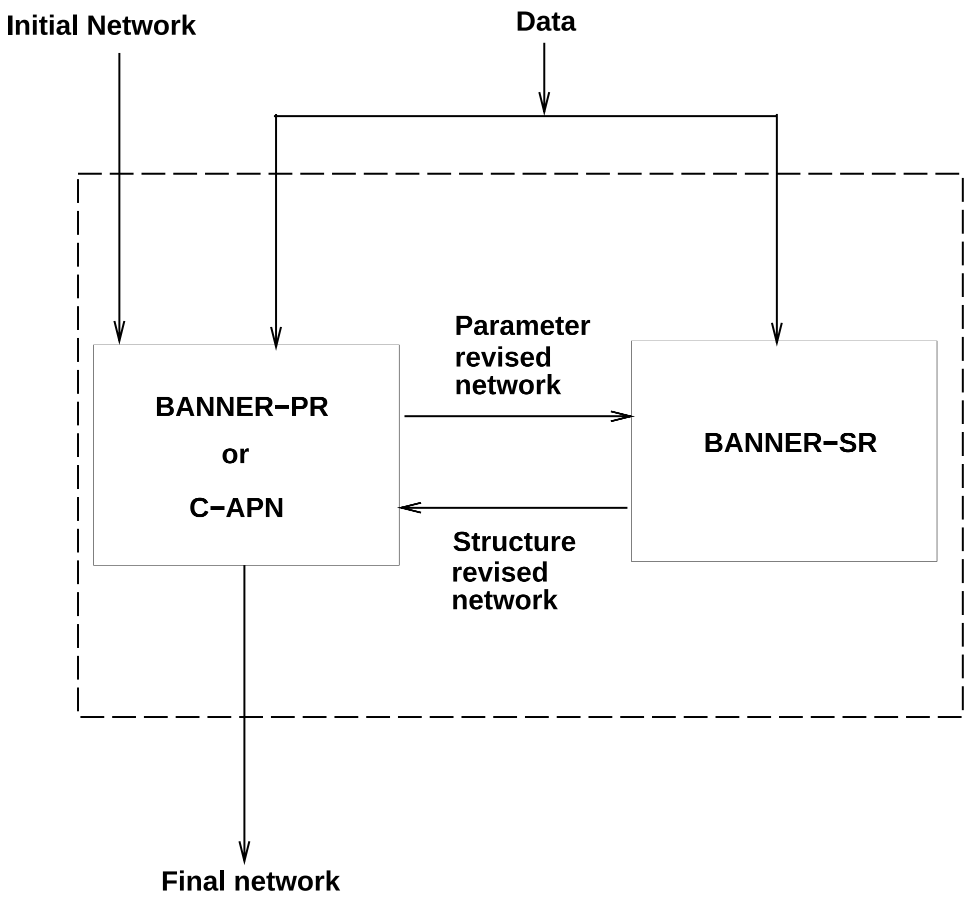


Figure 3.1: Overview of our system

some specific task such causal inference or abductive inference, or one may wish to build a general network that will perform well for any inference task.

Tables 3.1, 3.2, and 3.3 shows the applicability of BANNER-PR, and C-APN and BANNER-SR to various node types, network architectures, and inference types. A check mark ($\surd$) indicates that the algorithm is applicable to that case. We have introduced the new term, *virtual polytrees*, to describe networks that have loops such that each loop can be broken by an instantiated evidence variable. Such a network can be treated as a polytree for the purpose of inference.

Another factor that affects the applicability of these systems is the completeness of data. Data is said to be incomplete when there are hidden variables or when some of the examples in the data are missing values for some observed variables. Both C-APN and BANNER-SR can be used with incomplete data, as will be demonstrated in Chapter 6. BANNER-PR can handle incompleteness only under certain conditions. Recall that BANNER-PR can be used to revise the parameters of a network with loops only as long as all the loops in the network are broken by instantiated evidence variables. Thus, BANNER-PR can be applied to networks with hidden variables either when the hidden variables do not participate in any loop, or when they participate in loops that can be broken by other instantiated variables. Similarly, BANNER-PR can be applied to problems where some of the examples are missing values for some observed variables, as long as these variables are not required to break loops.

In summary, although BANNER-PR is the most restrictive algorithm, it is more efficient than C-APN and should be used whenever possible, i.e. in all cases where the network is a polytree or a *virtual polytree*, where the network is being trained for causal prediction, and where the data is either complete or satisfies the restrictions described above. We use C-APN for problems that do not fit these criteria.

Algorithm	Node Type	
	Unrestricted Models	Noisy-Or/And Models
BANNER-PR		$\surd$
APN	$\surd$	$\surd$
BANNER-SR		$\surd$

Table 3.1: Applicability of the algorithms to various node types

Algorithm	Architecture		
	Loops	Virtual Polytrees	Polytrees
BANNER-PR		$\surd$	$\surd$
APN	$\surd$	$\surd$	$\surd$
BANNER-SR	$\surd$	$\surd$	$\surd$

Table 3.2: Applicability of the algorithms to various network architectures

Algorithm	Type Of Inference			
	Causal	Abductive	combination	no particular task
Banner-Pr	√			
APN	√	√	√	√
Banner-Sr	√	√	√	√

Table 3.3: Applicability of the algorithms to various types of inference tasks

3.3 Converting from a Logical Theory to a Bayesian Network

The initial theory given to Banner may be in the form of a Bayesian network with Noisy-Or and Noisy-And nodes, or in the form of propositional Horn-clause rules, in which case the Horn-clause rules are converted into a Bayesian network with Noisy-Or and Noisy-And nodes.

The conversion is very straightforward. A consequent of a rule in the theory is converted into a Noisy-And node whose parents are the antecedents of the rule. When there are multiple rules with the same consequent, each rule is converted into a Noisy-And node, and then all of these Noisy-And nodes are, in turn, connected to a Noisy-Or node that maps the disjunction denoted by the multiple rules. A negation is handled by introducing a *negation node*, with the node corresponding to the proposition being negated as its parent. A negation node is a regular probabilistic node, whose conditional probability table is set so that it reverses the value of its parent. Since such a node is introduced for the specific purpose of negating the value of a variable, the parameters on the link between a negation node and its parent are never revised. Nor is the parent set of such a node, which includes just the node that is being negated, revised during structure revision.

Figure 3.3 shows the Bayesian network corresponding to the portion of a logical theory shown in Figure 3.2. Here, $Rule1$ has been converted into a Noisy-And node $G1$ that represents the consequent of the rule, *graspable*. The antecedents of the rule are the parents of this node. $Rule2$ is similarly represented by the Noisy-And node $G2$. The Noisy-Or node, *graspable*, maps the disjunction denoted by $Rule1$ and $Rule2$. The negation node $not\text{-}fragile$ represents the negated antecedent in $Rule1$, and is a regular, probabilistic node whose conditional probability table is pre-set to reverse the value of its parent, and is never revised. The sources of the DAG representing the Bayesian network, i.e. nodes *width-small*, *has-handle*, and *insulating* are regular probabilistic nodes. Finally, the probabilities on the links are all set to random values close to 1.0 to mimic the logical theory. It is necessary to perturb the weights around the value 1.0 to break the symmetry for better convergence while revising the parameters of the network.

Rule 1: graspable $\longleftarrow$ has−handle

Rule 2: graspable $\longleftarrow$ width−small, insulating , not fragile

Figure 3.2: Propositional Horn-clause theory

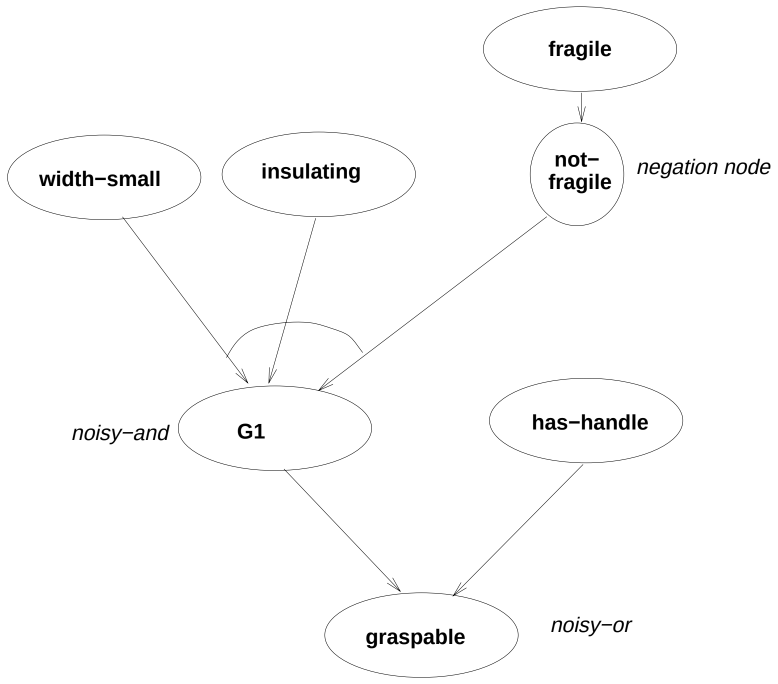


Figure 3.3: Bayesian network derived from a Horn-clause theory

Chapter 4

Parameter Revision: Learning Conditional Probabilities

As discussed earlier, the goal of this research is to design an efficient and effective algorithm for revising Bayesian networks composed of Noisy-Or/And nodes, in the presence of hidden variables. An essential component of such an approach is an efficient technique for revising the parameters of the network. In this chapter, we present BANNER-PR, a technique that revises the parameters of a network by using gradient descent to minimize the mean squared-error between the measured and the computed values of certain target variables.[1] In addition to being restricted to Noisy-Or/And nodes, it only works on problems where the inference is causal (i.e. from cause to effect). Thus, it can only learn networks whose sources are specified as evidence, and whose sinks are specified as target classes. Furthermore, it can only be applied to polytrees and *virtual polytrees*. The reason for these restrictions will become clear when we present the details of the algorithm.

As discussed in section 3.2, the structure revision algorithm is more general in its applicability than BANNER-PR. In order to evaluate our technique on more general learning scenarios, we have also implemented a variant of the *APN* algorithm (Russell et al., 1995) (Section 2.5) for revising parameters. This chapter will also discuss our extension to APN, called C-APN and present an experimental comparison of BANNER-PR, APN, and C-APN.

4.1 BANNER-PR

4.1.1 Overview of BANNER-PR

The learning algorithm used by BANNER-PR is analogous to the standard backpropagation algorithm used to train a multi-layered feedforward network (McClelland & Rumelhart, 1988). It uses gradient descent to minimize the mean-squared error between the measured and the computed values of certain target variables. The algorithm is as follows:

1. Initialize the parameters of the network either randomly or based on some prior knowledge.

2. For every example in the training data

[1]By adopting the standard technique of initially setting the parameters to random values, the technique can also learn parameter values from scratch.

(a) Place the evidence on the network and propagate beliefs through the network.

(b) Compute the mean squared-error at the target class nodes.

(c) Back-propagate the errors from the target nodes to the evidence nodes.

(d) Compute the gradient of the parameter at each link, based on the error associated with the node at the head of the link, and modify the parameter accordingly.

3. Repeat Step 2 for several cycles until a desired level of training performance is reached. Each cycle through the entire data set is called an *epoch*.

Section 3.3 discussed how the parameters of the network are initialized in Step 1 of the algorithm. Step 2a instantiates the network with the given evidence and propagates beliefs as discussed in Section 2.1.2. The following subsections will present the details of the backpropagation phase (steps 2c and 2d), followed by a discussion of the criteria used to decide when to stop training (step 3).

4.1.2 Backpropagation

Once the mean-squared error is computed for the nodes corresponding to the target class variables (henceforth called output nodes), it is propagated back through the network. The gradient to be applied to each of the parameters is computed based on these errors. The mean squared-error function is defined as:

$$E[\mathbf{w}] = 1/2 \sum_{i\mu} (\zeta^\mu(N_i = True) - Bel^\mu(N_i = True))^2 \tag{4.1}$$

where $\zeta^\mu(N_i = True)$ is the actual belief that the i-th target variable N_i is true for pattern μ of the data, and $Bel^\mu(N_i = True)$ is the predicted value of the belief that the i-th target variable is true for this same pattern.

The gradient is computed by incorporating the belief functions for Noisy-Or/And nodes (Equations 2.4 and 2.5) into the error function above and taking its partial derivative with respect to the parameters. This results in the following gradients for Noisy-Or and Noisy-And nodes respectively:

$$\Delta^\mu c_{ij} = \begin{cases} \eta\delta_j^\mu Bel^\mu(N_i = True) \prod_{k \neq i}(1 - c_{kj} Bel^\mu(N_k = True)) \\ -\eta\delta_j^\mu(1 - Bel^\mu(N_i = True) \prod_{k \neq i}(1 - c_{kj}(1 - Bel^\mu(N_k = True)) \end{cases} \tag{4.2}$$

where c_{ij} is the parameter on the link from node N_i to node N_j, η is the learning rate, and δ_j^μ is the error associated with node N_j for pattern μ, which, for output nodes, is defined as

$$\delta_j^\mu = \zeta^\mu(N_j = True) - Bel^\mu(N_j = True).$$

The error associated with an interior node is the sum of the errors propagated back from its children. The error propagated to N_i from its child N_j is:

$$\delta_{ji} = \begin{cases} -\delta_j c_{ij} \prod_{k \neq i}(1 - c_{kj} Bel(N_k = True)) & \textit{(for Noisy-Or)} \\ \delta_j c_{ij} \prod_{k \neq i}(1 - c_{kj}(1 - Bel(N_k = True)) & \textit{(for Noisy-And)} \end{cases} \tag{4.3}$$

where δ_j is the error at node N_j.

In addition to Noisy-Or/And nodes, the networks generated by BANNER could also have *negation* nodes, that model logical negations (see Section 3.3). These are regular, probabilistic nodes, with a single parent corresponding to the variable being negated. BANNER-PR computes the belief of a negated node according to:

$$Bel(N_j = True) = 1 - Bel(N_i = True),$$

where N_j is a negation node, and N_i is its sole parent. The error propagated from a negation node N_j to its parent N_i for pattern μ is given by

$$\delta_{ji}^{\mu} = -\delta_j^{\mu}.$$

The computations presented here and in Section 2.1.2 lead us to the reason why BANNER-PR is limited to causal inference and allows for only certain kinds of loops. Since the gradients are based on Equations 2.4 and 2.5, which were derived based on the assumption that the inference is causal, they would not be appropriate for non-causal inference. Moreover, the functions for forward propagation shown above are no longer applicable in the presence of loops, unless the evidence renders the network to be a *virtual polytree.* Removing these restrictions is an important direction for future research and is discussed in Chapter 8.

4.1.3 Stopping Criteria

Knowing when to stop training is one of the crucial aspects of a gradient descent training algorithm. If the network is not trained enough, then it will not learn the training data effectively and hence may not be able to generalize well to unseen cases. If it is trained for too long, it may overfit the training data and therefore lose out on generalization. In order to avoid overfitting, we use 10-fold internal cross-validation to determine when to stop training. Thus, each set of training data is partitioned into 10 equal subsets. Ten networks are trained for a pre-specified number of cycles, each using 9 of the subsets for training and one for testing. The generalization performance of each network on the internal test sets is monitored at the end of each training cycle, and the average generalization error is computed for each cycle. To avoid overfitting, the number of training epochs that shows the least average generalization error is picked as the number of epochs needed to train the network on the full training set. In addition, when this number is less than the pre-specified number of training cycles, and when BANNER-PR determines that training the network on the full training set for the pre-specified number of cycles would lead to 100% training accuracy, it sends signal to the overall BANNER algorithm that the current structure overfits the training data.

4.2 APN and C-APN

Section 2.5 describes APN (Russell et al., 1995), a technique for revising the parameters of a network of a given structure using gradient descent search. This technique is more general in its applicability than BANNER-PR in that it is not restricted to Noisy-Or/And distributions and can be applied to networks with loops. Recall that APN uses gradient descent to optimize $\ln P_w(D|B)$, i.e. the log of the probability assigned by network B to data D when the parameters are set to

w. The algorithm iterates over each example in the data set several times until convergence. For each example, it instantiates the network with the observed values which serve as evidence, and propagates the evidence throughout the network. It then updates the parameters of the network, using the following equation to compute the gradients:

$$\frac{\partial \ln P_w(D)}{\partial w_{ijk}} = \sum_{l=1}^{m} \frac{P_w(x_{ij}, u_{ik} \mid D_l)}{w_{ijk}} \tag{4.4}$$

where m is the number of training instances, x_{ij} is the jth possible assignment to variable X_i, u_{ik} is the kth possible value assignment to the parents of X_i, w_{ijk} is the probability that variable X_i takes on its jth possible assignment given that its parents U_i take on their kth possible assignment. $P_w(x_{ij}, u_{ik} \mid D_l)$ is computed as a by-product of inferring the probabilities of the variables in the network, given the evidence in D_l. Note that there is no backpropagation of errors. All the information for computing the gradient of the parameter at a link is available at the node at the head of the link and the parents of that node.

Although the authors do not report any results on Noisy-Or models, they point out how their technique can be extended to compute gradients for Noisy-Or parameters. Applying the chain rule to Equation 4.4, and noting that all Noisy-Or/And variables are binary-valued, we get

$$\begin{aligned} \frac{\partial \ln P_w(D)}{\partial q_{ji}} &= \textstyle\sum_{l=1}^{m} \sum_k \displaystyle\frac{P_w(x_i = false, u_{ik} \mid D_l)}{w_{i0k}} \times \frac{\partial w_{i0k}}{\partial q_{ji}} + & (4.5) \\ & \qquad \frac{P_w(x_i = true, u_{ik} \mid D_l)}{w_{i1k}} \times \frac{\partial w_{i1k}}{\partial q_{ji}} & (4.6) \end{aligned}$$

where q_{ji} is the parameter on the link from node X_j to node X_i, w_{i0k} is the probability that variable X_i is $false$ given that its parents U_i take on their kth possible assignment, and w_{i1k} is the probability that variable X_i is $true$ given that its parents U_i take on their kth possible assignment. The interior sum ranges over all possible instantiations of the parents of variable X_i. Further derivation leads to the following gradient for Noisy-Or nodes:

$$\begin{aligned} \frac{\partial \ln P_w(D)}{\partial q_{ij}} &= \textstyle\sum_{l=1}^{m} \sum_{k_i = k_{j=true}} \prod_{i=T_k - j} \quad q_{il} \times \displaystyle\frac{P_w(x_i = false, u_{ik_i} \mid D_l)}{w_{i0k_i}} - \\ & \qquad q_{il} \times \frac{P_w(x_i = true, u_{ik_i} \mid D_l)}{w_{i1k_i}} \end{aligned} \tag{4.7}$$

where $k_{j=true}$ is an instantiation of the parents of variable X_i where X_j is true, and T_k is the set of parents of X_i that are $true$, The equation for the gradient for a Noisy-And node is

$$\begin{aligned} \frac{\partial \ln P_w(D)}{\partial q_{ij}} &= \textstyle\sum_{l=1}^{m} \sum_{k_i = k_{j=false}} \prod_{i=F_k - j} \quad q_{il} \times (\displaystyle\frac{P_w(x_i = true, u_{ik_i} \mid D_l)}{w_{i1k_i}} - \\ & \qquad q_{il} \times \frac{P_w(x_i = false, u_{ik_i} \mid D_l)}{w_{i0k_i}} \end{aligned} \tag{4.8}$$

where $k_{j=false}$ is an instantiation of the parents of variable X_i where X_j is $false$, F_k is the set of parents of X_i that are $false$, w_{i0k} is the probability that variable X_i is $false$ given that its parents U_i take on their kth possible assignment, and w_{i1k} is the probability that variable X_i is $true$ given that its parents U_i take on their kth possible assignment.

While APN optimizes the probability assigned by the network to the data as a whole, sometimes it may be desirable to learn a network that is optimal for its intended task. Specifically, when the Bayesian network is being trained to be a classifier, it is desirable to attempt to optimize its *classification* accuracy. Friedman and Goldszmidt (1996) have argued in favor of learning a network that best estimates the probability distribution of the class variables conditioned on the evidence or attributes. Therefore, the function to be optimized is $\ln P_w(C \mid E)$, which is the log of the conditional distribution of certain class variables C, conditioned on some evidence E. In order to see how APN can be used to optimize this metric, note that by applying Bayes law we get

$$\ln P_w(C \mid E) = \sum_{l=1}^{m} \ln P_w(C_l \mid E_l) \tag{4.9}$$

$$= \sum_{l=1}^{m} (\ln P_w(C_l, E_l) - \ln P_w(E_l)) \tag{4.10}$$

where m is the number of training instances. Therefore,

$$\frac{\partial \ln P_w(C \mid E)}{\partial w_{ijk}} = \sum_{l=1}^{m} (\frac{\partial \ln P_w(C_l)}{\partial w_{ijk}} - \frac{\partial \ln P_w(E_l)}{\partial w_{ijk}}) \tag{4.11}$$

Thus, the gradient to optimize $\ln P_w(C \mid E)$ can be computed as the difference between the gradient computed wrt. $\ln P_w(C, E)$ using APN (as discussed in section 2.5) and the gradient computed wrt. $\ln P_w(E)$ using APN. This is the procedure adopted by C-APN, and since it is based on APN, it can handle Noisy-Or, Noisy-And, as well as more general models of interactions.

The techniques described above are similar to Banner-Pr in that they use gradient descent to learn the parameters of a network. In fact, it has been shown that the criterion of minimizing the mean-squared error between the observed values of the target variables and their predicted values, as used by Banner-Pr, is equivalent to the criterion, as used by C-APN, of finding a network that best explains the conditional distribution of the target variables (Rumelhart, Durbin, Golden, & Chauvin, 1995; Opitz, 1995a). However, the computation of gradients for APN and C-APN involves explicit computation of the conditional probabilities tables for the nodes in the network. As discussed earlier, the size of the conditional probability table associated with a node is exponential in its fan-in, and one of the prime reasons for using Noisy-Or/And nodes is to circumvent the need for such tables. Using APN and C-APN, thus, nullifies the computational advantages of using Noisy-OR/And models. On the other hand, by restricting itself to certain kinds of network structures, Banner-Pr is able to directly exploit the computations involved in the Bayesian inference process, which in turn exploit the fact that Noisy-Or/And parameters combine linearly. The number of computations required by Banner-Pr for each training cycle is polynomial in the fan-in of the nodes, whereas the number of computations required by APN and C-APN for each training cycle is exponential in the fan-in of the nodes. Thus, although Banner-Pr and C-APN perform equivalent searches, the computational efficiency of the former far exceeds that of the latter.

Due to their ability to handle a wider range of network architectures and distributions, we have included APN and C-APN in the suite of parameter revision techniques that can used by Banner. Our implementation of APN and C-APN is built on top of a system for building and reasoning with Bayesian networks called IDEAL (Srinivas & Breese, 1993), and can handle Noisy-Or and Noisy-And nodes in addition to unrestricted models of interaction of influences.[2]

4.3 Experimental Evaluation

The experiment described in this section was performed to compare the effectiveness of APN, C-APN, and BANNER-PR in revising the parameters of a given network. Chapter 6 presents a more detailed evaluation of BANNER as a whole, which integrates both the structure revision component and the parameter revision component. The data set used in the experiment described here has also been used to evaluate the full system, and will be described again in detail in Chapter 6. Here, we give a brief description of the data set and discuss the performances of the three parameter revision techniques.

The standard practice in the field of learning Bayesian networks is to generate data from a pre-specified network and then try to use the data to re-learn the original network. The performance of the learning technique is evaluated by directly comparing the learned network to the original. However, in real-world applications the "correct" network is not known. Recently, there has been growing interest in using evaluation functions such as classification accuracy to demonstrate the effectiveness of techniques for learning Bayesian networks (Friedman & Goldszmidt, 1996; Russell et al., 1995). We use a real-world classification problem of DNA promoter recognition (Towell et al., 1990) to evaluate the learning algorithms. In our experiment, we follow the standard methodology used to evaluate machine learning methods, i.e. that of studying the effect of training by determining the classification accuracy of the trained network on a separate test set.

4.3.1 DNA Promoter Recognition

Given a strand of DNA, geneticists are interested in identifying those sequences of nucleotides that form a gene. This task is complicated by the fact that gene sequences are separated by sequences of nucleotides that do not encode any useful information. Thus, it is important to have techniques for identifying those portions of the DNA strand that encode genetic material. It has been observed that gene sequences are always immediately preceded by particular sequences of nucleotides called *promoters*. Thus, identifying *promoter* sequences would help identify the start of gene sequences. This can be viewed as a classification problem, where the task is to classify a sequence of nucleotides according to whether or not it is a *promoter*. Since geneticists have not yet formulated the rules classifying promoters accurately, there has been a lot of interest in learning such classifiers from data. Through analysis of biological literature of O'Neill and Chiafari (1989), Noordewier et al. (1991) have proposed a set of rules for recognizing *promoters* given a sequence of nucleotides. However, this theory is overly specific and classifies all promoter sequences as non-promoters, making it an ideal candidate for theory refinement.

The data set used for this evaluation consists of 106 examples, of which 53 are positive examples of a *promoter*, and 53 are negative examples. Each example has 57 input features, which represent nucleotides, each of which can take on one of four values, A, G, T or C. The input features represent a window over a DNA strand. The information in this window is used to determine the

[2]When contacted, the authors of APN were unable to release their implementation.

presence of a promoter. Each example is labeled according to whether or not it is a promoter. Figure 4.1 shows the initial logical theory for recognizing a DNA promoter sequence. Since APN and C-APN cannot handle networks with large large fan-ins, we used a slightly modified version of this theory for these experiments, where the last eight antecedents of the first rule for the *conformation* were deleted. Figure 4.2 shows a portion of the Bayesian network derived from this theory, using the procedure described in Section 3.3.

promoter	⟵	contact, conformation
contact	⟵	minus_35, minus_10
minus_35	⟵	(p-37 c) (p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)
minus_35	⟵	(p-36 t) (p-35 t) (p-34 g) (p-32 c) (p-31 a)
minus_35	⟵	(p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c) (p-31 a)
minus_35	⟵	(p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)
minus_10	⟵	(p-14 t) (p-13 a) (p-12 t) (p-11 a) (p-10 a) (p-9 t)
minus_10	⟵	(p-13 t) (p-12 a) (p-10 a) (p-8 t)
minus_10	⟵	(p-13 t) (p-12 a) (p-11 t) (p-10 a) (p-9 a) (p-8 t)
minus_10	⟵	(p-12 t) (p-11 a) (p-7 t)
conformation	⟵	(p-18 t) (p-5 c) (p-2 c) (p-4 c) (p-40 a) (p-47 c) (p-42 t) (p-43 t)
		(p-39 c) (p-22 g) (p-16 c) (p-8 g) (p-7 c)
		(p-6 g) (p-1 c) (p-46 a) (p-45 a)
conformation	⟵	(p-45 a) (p-44 a) (p-41 a)
conformation	⟵	(p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t) (p-16 t)
		(p-15 g) (p-1 a)
conformation	⟵	(p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t) (p-21 a)
		(p-20 a) (p-17 t) (p-15 t) (p-4 t)

Figure 4.1: DNA Promoter Recognition - Initial Domain Theory

The experiments were performed using the *re-sampling* methodology, where we conducted a series of 10 trials for each training set size. During each trial, the data set was randomly partitioned into a training set of a particular size, and a test set which consisted of the remaining examples in the data set. Each learning algorithm was trained on the training set and evaluated on the test set, and the results averaged over the 10 trials. As mentioned earlier, Banner-Pr used 10-fold internal cross-validation on the training set to determine the stopping point for each train-test split. However, the long training times required by APN and C-APN on this problem made it infeasible to use this approach to determine the stopping point for these techniques. For instance, it took 15 hours (real-time) for C-APN to train a network for 300 epochs on 90 examples on an UltraSparc I. Therefore, for these two techniques, all networks were trained to 300 epochs or until the error showed no improvement over 25 consecutive iterations, which ever was sooner.

There are two issues that we hoped to address with our experiment. The first issue is: given a problem to be solved using a Bayesian network, which of the three techniques produces better generalization results? Since the three algorithms search through equivalent search spaces, it is to be expected that they will perform comparably. However, we would expect that Banner-Pr would converge faster than the other algorithms because it learns the Noisy-Or/And parameters directly whereas the other algorithms learn the Noisy-Or/And parameters indirectly via the exponentially large conditional probability tables associated with each node. Figure 4.3 shows that Banner is

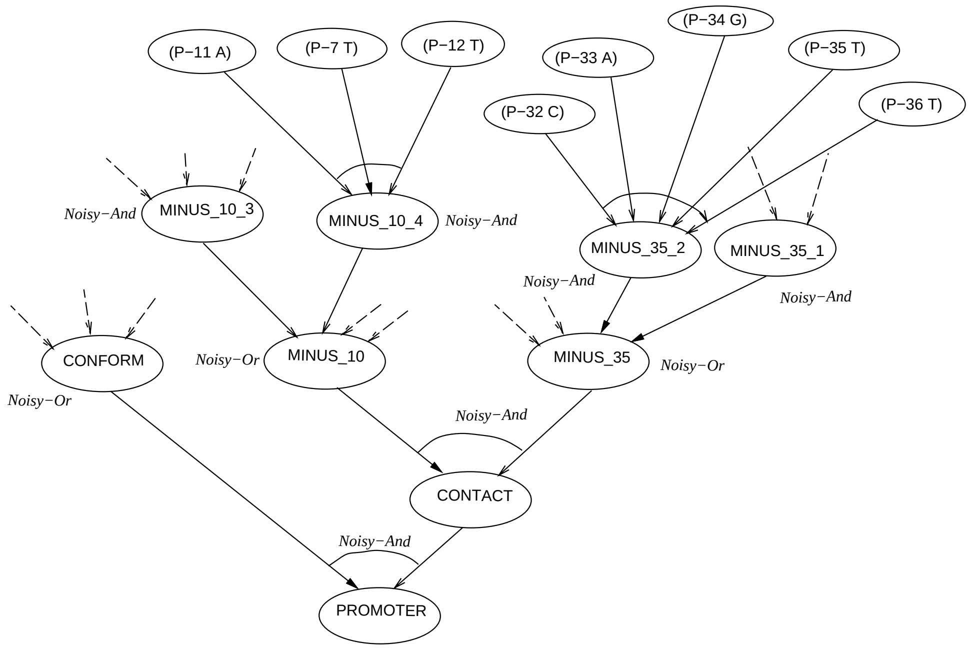


Figure 4.2: DNA Promoter Recognition - Bayesian Network

successful in learning networks with high classification accuracy, and performs better than APN or C-APN by large margins, except for very small training sets. The differences between the performances of BANNER and C-APN are statistically significant at the 0.001 level for training set sizes 20 and 60, and at the 0.01 level for training sets of size 40. The differences in performance between BANNER and APN is statistically significant at the 0.01 level for training set sizes of 40 and 60, and at the 0.001 level when the size of the training set is 20. The differences between APN and C-APN are not statistically significant.

It is clear from figure 4.4, which shows the accuracies of the three systems on the training set, that the poor performance of APN and C-APN on the test data is largely due to the fact that APN and C-APN haven't converged on the training data. An obvious solution is to let the networks train longer until they converge on the training data. However, both APN and C-APN were found to be very slow, which made it infeasible to train the networks longer. As mentioned earlier, it took 15 hours (real-time) for C-APN to train a network for 300 epochs on 90 examples on an UltraSparc I, whereas it only took 90 seconds for BANNER-PR to do the same. Of course, these algorithms are slowed down to some extent by the underlying Bayesian network simulator used by our implementation. However, real time aside, BANNER-PR was also found to converge with many fewer epochs than APN or C-APN. For instance, BANNER-PR needed an average of 60 training cycles to converge on 90 examples, whereas APN and C-APN could achieve less than 90% training accuracy even when trained up to an average of 200 cycles.

The second issue is whether or not it is better to train a network specifically for classification in order to get better classification performance. The graph indicates that C-APN is better than APN for most points in the learning curve. Although the differences in their performances are not statistically significant, they are encouraging enough to warrant further exploration of this issue by comparing the performances of these two techniques on more data sets. It would also be interesting to study the relative performances of these techniques when used to train networks for tasks involving diagnostic prediction or some combination of causal and diagnostic prediction.

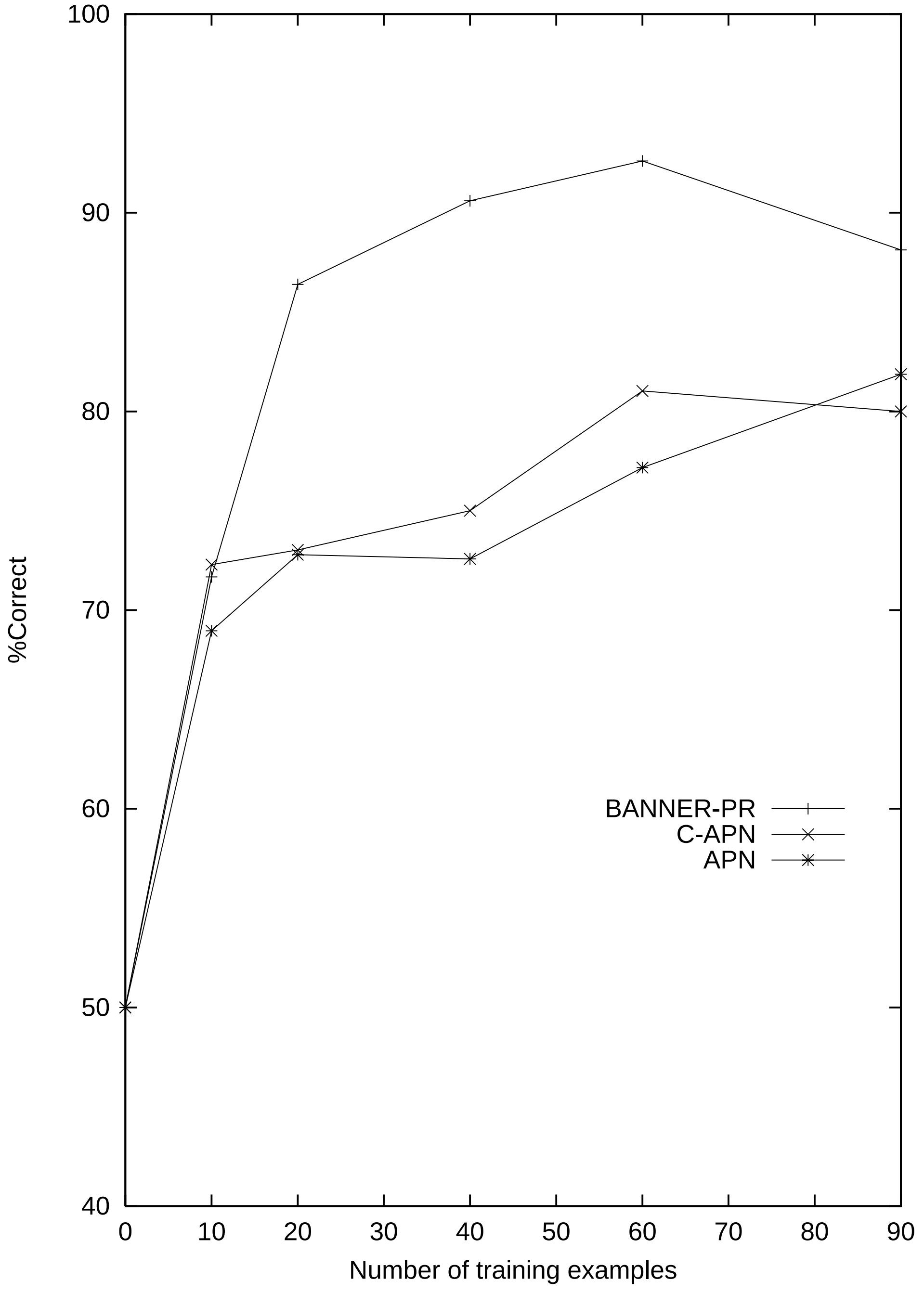


Figure 4.3: DNA Promoter Recognition: Performance of various parameter revision algorithms

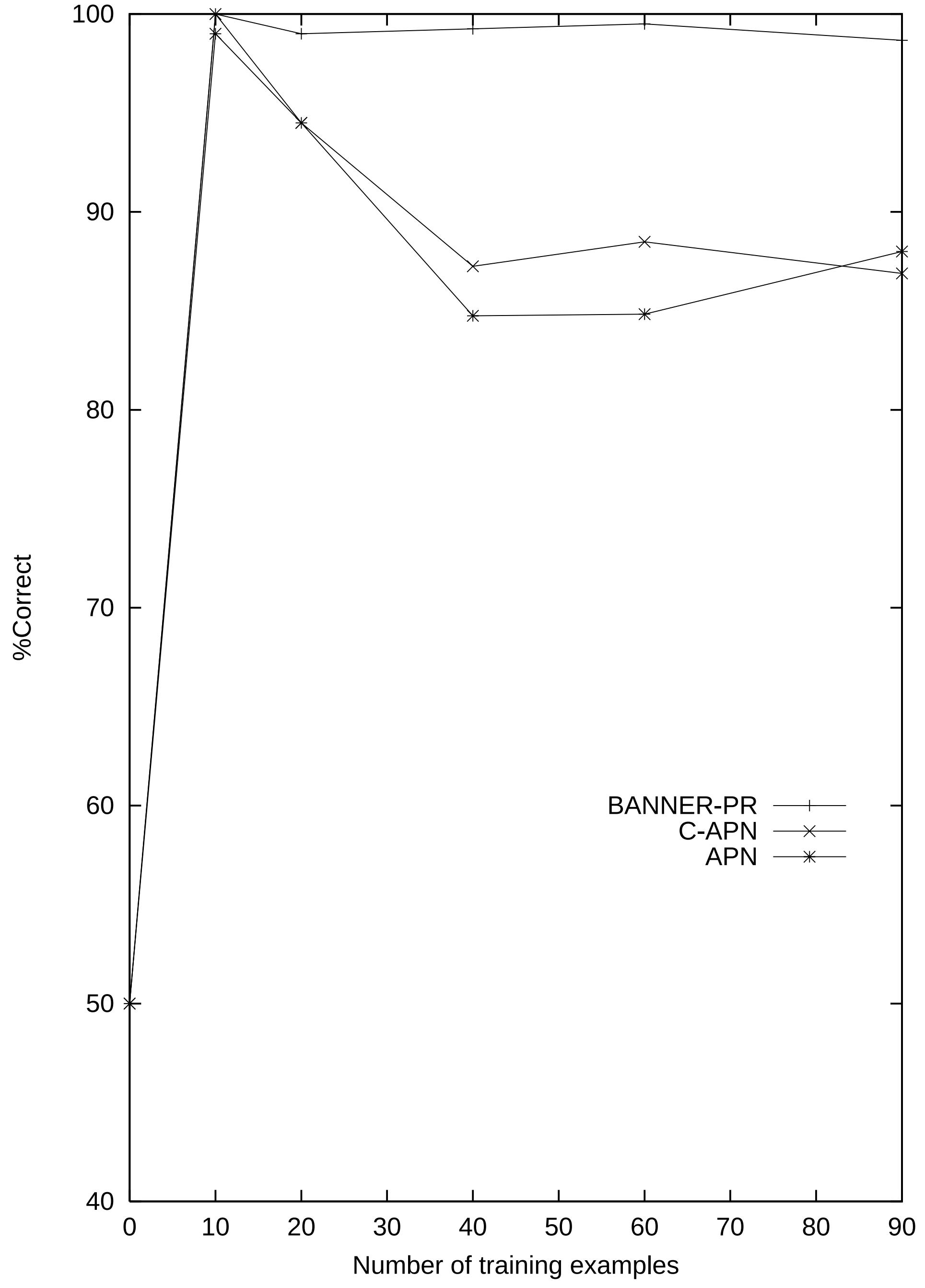


Figure 4.4: DNA Promoter Recognition: Accuracy on the training data

Chapter 5

Structure Revision

In designing an algorithm for revising the structure of a Bayesian network, we are faced with the following issues: *when*, *where* and *how* should a network be revised, and when to *stop* revising the network. The answer to the question of when the network should be revised is simple. Since the goal of refinement is to improve classification accuracy, a network should be revised when it fails to classify some data accurately. Our techniques for addressing the issues of where and how to revise the network form the backbone of our structure revision algorithm, and will be the main focus of this chapter. The success of a theory refinement algorithm depends on its ability to stop revising the theory before it overfits the training data. Section 5.5 will discuss this issue further.

Figure 5.1 shows a high-level outline of the algorithm. Given a set of examples consisting of observations of the evidence and the target variables, and an initial network, BANNER first uses the parameter revision module to revise the parameters of the network and fit the data. If the network still misclassifies some examples, BANNER invokes the structure revision module, which uses the examples to determine the portions of the network to be revised in order to correct the misclassifications. Some of these revisions are implemented and the network is retrained to re-calibrate the parameters. These steps are repeated until further revision fails to improve the classification accuracy of the network on the data.

Given: An initial approximate theory in the form of a Bayesian network with Noisy-Or/And nodes, and a set of training data.
Output: A revised network that classifies the data accurately.
Algorithm:

```
1. Initialize the parameters of the network either randomly or based on some prior knowledge.
2. Repeat steps a and b until a desired level of performance is reached.
    (a) Train the network to revise the parameters.
    (b) If any of the examples are misclassified
        i. Identify portions of the network to be revised.
        ii. Revise the network structure at these points so as to cover the misclassified examples.
```

Figure 5.1: Outline of the Algorithm

The purpose of the structure revision module is to repair a theory when it fails to classify examples accurately. One approach to repairing a theory, employed by most existing techniques for learning Bayesian networks (Friedman, 1997; Singh, 1997; Kwoh & Gillies, 1996; Cooper & Herskovits, 1992; Ramoni & Sebastiani, 1997), is to consider all possible single revisions and select the one that leads to the greatest improvement in accuracy. Since there are usually several possible single revisions to a network, and only a small fraction of these are needed to fit the training data, such *generate then test* approaches (Mitchell, 1997) spend considerable computational effort in exploring unproductive paths in the search space. An alternative is the *example-driven* approach, often used by logical theory refinement algorithms (Richards & Mooney, 1995; Ourston & Mooney, 1994; Opitz & Shavlik, 1993), where the data is used to attribute failures in prediction to specific portions of the network. This restricts the search space considerably by focusing attention on a subset of variables that can be held responsible for an erroneous prediction by the network. BANNER is based on the idea that, since we are considering networks with Noisy-Or and Noisy-And nodes which are analogous to logical disjunction and conjunction, techniques similar to those used for revising logical theories can be applied to revise these networks.

Consider the reasoning used in diagnosing the cause of misclassifications arising from a logical theory. Figure 5.2 shows a logical theory in the form of Horn-clauses, and a couple of examples that are misclassified by this theory. The theory classifies the first example, labeled a positive example of H, as a negative example, indicating that the theory is too specific. This could be because

1. there should be more rules for concluding H, or
2. one or more of the rules for concluding H have more antecedents than necessary, or
3. the rules for X or Y are too specific, i.e. they have more antecedents than necessary, or
4. there should be more rules for concluding X or Y.

Theory:

$H \leftarrow X, F.$
$H \leftarrow Y, G.$
$X \leftarrow A, B.$
$Y \leftarrow C, D.$

Examples:

1. G = True, A = False, B = False, C = False, D = False, E = True , F = False , H=True
2. G = False, A = True, B = True, C = False, D = True, E = False, F = True , H=False

Figure 5.2: Example of an incorrect logical theory

The second example, labeled a negative example of H, is classified as a positive example by the theory. This means that the theory is too general, which could be because

1. there are too many ways of concluding H, i.e. one of the rules for concluding H should be deleted, or
2. the rules for concluding H are too general and need additional antecedents, or

3. there are more rules than necessary for concluding X or Y, or

4. the rules for concluding either of X or Y are too general and need additional antecedents.

These observations not only diagnose the errors in the theory, they also suggest ways in which the theory could be revised to correct these errors. Most theory refinement systems use a combination of abduction and induction to effect such reasoning as described above (Mooney, 1997). For each positive example not provable by the theory, these systems use abduction to generate a small set of assumptions that would allow the example to be proved. These assumptions are then used to suggest modifications to the theory. For negative examples proved by the theory, these techniques use induction to learn new antecedents for one or more rules that render these examples unprovable. For example, given the first failing example shown in figure 5.2, these techniques would generate $Y = True$ as a minimal assumption that proves the example. Thus, possible repairs to the theory would be to delete Y from the second rule, or to learn a new rule for proving Y. This is similar to the reasoning outlined above to explain this misclassification, with the added constraint of generating a minimal fault hypothesis. Given the second failing example, these techniques would learn new antecedents for the first or the third rule in the theory. Note that, since in general, the problem of generating a minimal fault hypothesis is intractable, these techniques use a greedy algorithm to generate an approximately minimal fault hypothesis.

Our goal is to develop similar insights into diagnosing faults in Bayesian networks. Figure 5.3 shows a Bayesian network with Noisy-Or and Noisy-And nodes, and an example that it misclassifies. Assume that the parameters of the network have been set so that the behavior of the nodes is similar to that of their logical counterparts. Since the degree of belief in G, given the input features in the example, is less than 0.5, this example is misclassified as a negative example of G. This could be because

1. the Noisy-Or node G is missing a parent which would provide the necessary positive evidence, or

2. the Noisy-And nodes E and F have unnecessary parents that are providing too much negative evidence, or

3. some of the parents of E and F are themselves too specific, and should be made more general by interspersing the respective links with a Noisy-Or node with attached enabling influences.

Notice that this reasoning is analogous to the reasoning used to propagate blame for misclassification through a logical theory. The main difference here is that the degree of blame passed from a node to its neighbors should now be proportional to the parameters associated with the links between the nodes. Moreover, the weights on the links reflect, to some extent, the over-generality or over-specificity of the theory and have to be taken into account while assigning blame. Finally, the *blame* factor itself is different for the two cases. For logical theories, *blame* simplify reflects misclassification, whereas for Bayesian networks, *blame* also reflects the amount by which the belief in the truth of any particular node should change in order to correct the misclassification.

So now, we need to design an algorithm for diagnosing the cause of failures in a Bayesian network that incorporates the reasoning outlined above, while taking into account quantitative

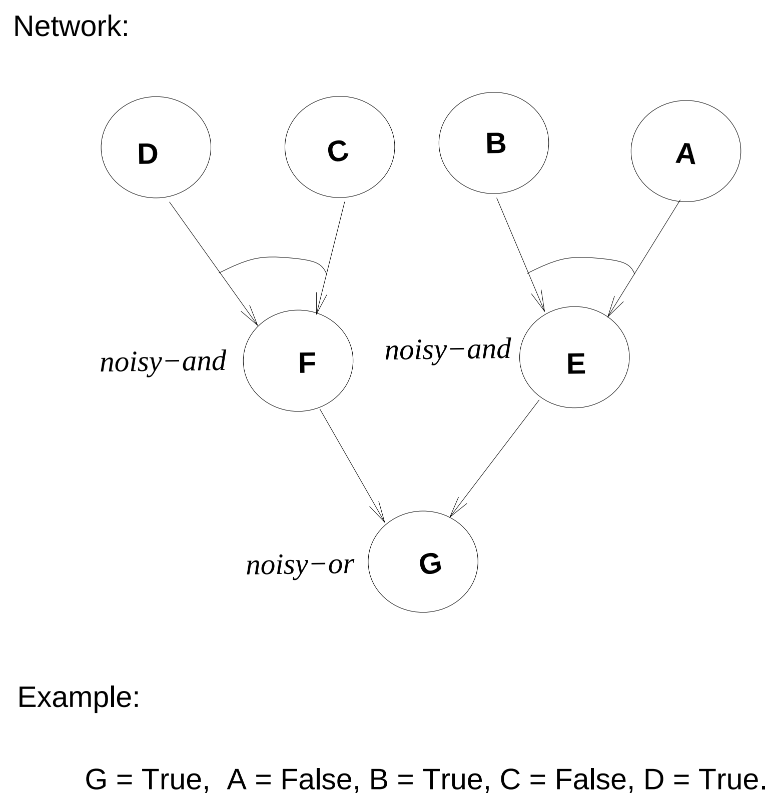


Figure 5.3: Example of an incorrect Bayesian network

information such as the network parameters and the belief associated with each node. Consider the following observations:

1. Typically a Bayesian network is used to infer the probability of some variables given some evidence. Thus, we already have algorithms that propagate beliefs through networks, taking into account all the quantitative information. Our algorithm is designed to use these inference mechanisms to assign blame to different parts of the network.

2. *Leak* nodes, as described in Section 2.1.2, provide a way to incorporate the kind of reasoning outlined above into the Bayesian network. For example, adding a leak node to a Noisy-Or node provides a way of encoding incompleteness in the knowledge of all the causes of the node. This leak node could be used to signal a situation where none of the current causes of the node can explain its target outcome. This is described in greater detail in the next section.

These observations lie at the heart of our structure revision algorithm which diagnoses faults in a Bayesian network by temporarily instrumenting the network at each node with *leak* nodes which serve as indicators of possible repairs to the theory. Individual examples are then used to select a small set of these repairs to be implemented. In the following two sections we present the details of how the network is augmented with leak nodes, and then discuss the procedure for using these leak nodes to select revision points.

5.1 Augmenting the Network with Leak Nodes

The procedure for augmenting a network with leak nodes is best illustrated with an example. Figure 5.4 shows the network from figure 5.3 augmented as follows:

- Each Noisy-Or and Noisy-And node in the network has an additional node added to its set of parents. We will refer to these nodes as *node-leak* nodes. The parameter on the link between the node and its new parent is set to be very low (0.01), in order to simulate a logical disjunction. The augmentation is not complete until the prior probabilities of the newly introduced *node-leak* nodes are set. Since these leak nodes represent possible repairs to the theory, which is assumed to be correct unless there are indications to the contrary, they are initially set to have a low prior probability. However, when the algorithm detects misclassifications, it estimates these prior probabilities by training a copy of the network, augmented with *node-leak* nodes, using the parameter revision module. Figure 5.5 illustrates this in greater detail.

- The Noisy-Or nodes have each of their parents routed through an *intervening* Noisy-And node, and the Noisy-And nodes have each of their parents routed through an intervening Noisy-Or node. The intervening nodes themselves have attached leak nodes, henceforth referred to as a *link-leak* nodes. The intervening nodes and the *link-leak* nodes together augments the links in the network. For example, the connections to the Noisy-Or node G have been modified so that its parent E is now connected to an intervening Noisy-And node G-E, which is then connected to G. The parents of node G-E are the nodes E, and the *link-leak* node G-E-*leak*. Notice that this transformation is similar to the conceptual view of a Noisy-Or node as described in Section 2.1.2. In fact, the two would be equivalent if the weights on the links are set to simulate logical conjunctions and disjunctions and the prior probability of the *link-leak* node is set to the weight on the link between nodes E and G in the unaugmented network. Setting the weights in this manner completes the augmentation of node G. Other Noisy-Or and Noisy-And nodes in the network are similarly transformed.

Once a network has been augmented in this manner, the stage is set for blame assignment, i.e. the process of determining the contribution of each node and link in the network to the failures in prediction.

5.2 Blame Assignment

The leak nodes introduced into the network, as described above, represent possible faults in the theory. Once these nodes are in place, BANNER performs abduction on each misclassified example to generate a set of repairs that would lead to the example being classified correctly. Note that this use of abduction is very similar to the way that several logical theory refinement algorithms use abduction to generate revision points in the theory (Ourston & Mooney, 1990, 1991, 1994; Wogulis & Pazzani, 1993; Wogulis, 1994; Baffes, 1994; Baffes & Mooney, 1996; Brunk, 1996). However, logical theories require special algorithms for abduction, whereas for Bayesian networks, this process simply involves instantiating both the evidence and the target variables to their observed values and inferring the beliefs associated with the leak nodes using standard Bayesian inference algorithms.

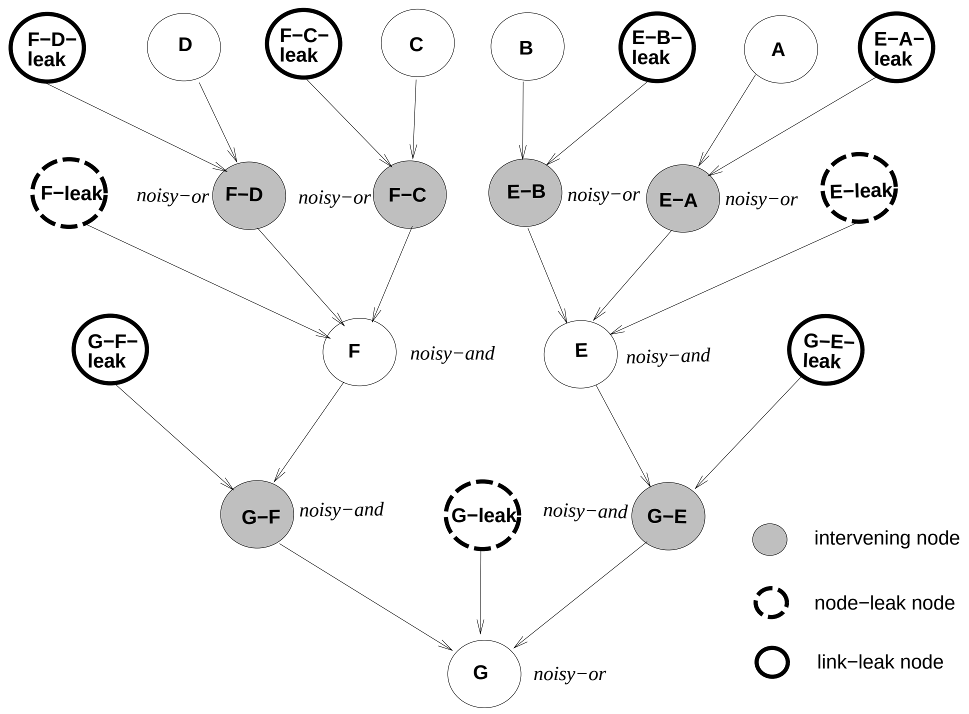


Figure 5.4: Augmenting a network with leak nodes

1. set *leak-net* = *train-net* augmented with *node-leak* nodes, where *train-net* is the current version of the network being trained.
2. Train network *train-net* to revise parameters.
3. If there are some misclassified examples
    (a) Train network *leak-net* to estimate prior probabilities of the *leak* nodes.
    (b) Set *augmented-net* = *train-net* augmented with *node-leak* and *link-leak* nodes.
    (c) Copy priors of *node-leak* nodes from *leak-net* to *augmented-net*.

Figure 5.5: Procedure for estimating prior probabilities of *node-leak* nodes

Another difference is that BANNER uses abduction to generate repairs for all the misclassified examples, while the other algorithms do so only for failing positive examples. Finally, whereas other techniques can narrow down the blame to particular rules, the leak nodes used here enable BANNER to narrow down blame further to specific links in the network, which for logical theories, is equivalent to blaming particular antecedents of a rule.

Thus, for each misclassified example, BANNER performs inference on the network augmented with leak nodes to collect a set of *node-leak* nodes and *link-leak* nodes, whose beliefs deviate from their prior probability by more than a certain amount[1]. Such leak nodes are said to *cover* the example, and indicate potential revision points in the theory. When the belief in the truth of a leak node decreases from its prior, the node is said to act as an *inhibitor.* Similarly, when the belief in the truth of a leak node increases from its prior, then the node is said to act as an *enabler.* The purpose of this distinction will be explained in Section 5.4. Each leak node covering an example is associated with the degree to which its belief deviated from its prior belief. This indicates the extent to which each of the leak nodes is to be blamed for the misclassification. Once leak nodes are collected for all of the misclassified examples, BANNER uses a greedy set covering algorithm, where the contribution of each leak node is weighted by its degree, to generate a small set of leak nodes that cover all of the misclassified examples. Figure 5.6 gives the details of this algorithm, which takes as input a set of misclassified examples, ME, and a set of leak nodes, LN, and returns *revision-points*, a small set of leak nodes that covers all the misclassified examples. The function *degree*(L, E) returns the degree of deviation in belief exhibited by the leak node L for example E, and *argmax* $f(x)$ returns the value of x for which the function $f(x)$ is maximized.

```
greedy-set-cover($ME$,$LN$)
    1. Set revision-points = empty
    2. Repeat until $ME$ is empty:
        (a) For each leak-node $L \in LN$, set $blame(L) = \sum_{j \in ME_L} degree(L,j)$, where
            $ME_L$ = set of misclassified examples covered by $L$.
        (b) Set $L_{max} = argmax_{l \in LN}\ blame(l)$.
        (c) Add $L_{max}$ to revision-points.
        (d) Remove the set of misclassified examples covered by $L_{max}$ from $ME$.
    3. Return revision-points.
```

Figure 5.6: Greedy Set Cover Algorithm

While BANNER uses only the misclassified examples to generate a set of revision points, it performs inference on the augmented network for all the examples in the training data, including the ones that were classified correctly. Thus, it generates a list of *node-leak* nodes and *link-leak* nodes that are *enabled* or *inhibited* for each example in the training data. The purpose of this will become clearer in the next section which discusses the operators used by BANNER to repair the theory.

[1]This threshold amount was set to be equal 10% of the prior probability, a criterion that was heuristically determined.

5.3 Revision Operators

Having addressed the issue of deciding where the network should be revised, we now turn our attention to the issue of how it should be revised. Recall that BANNER uses the augmented network to pick *node-leak* and *link-leak* nodes that form the set of revision points. Depending on whether a revision point is a *node-leak* node, or a *link-leak* node, BANNER implements one of the following three revision operators:

1. **Adding a new parent:** When a candidate revision point is a *node-leak* node, BANNER adds a new parent to the node in the original network corresponding to the node in the augmented network to which the leak node is attached. This new parent may be any node in the network that is not already a parent or a descendant of the node being revised (in order to prevent directed cycles). For instance, suppose that the *node-leak* node, *G-leak*, in the augmented network shown in figure 5.4 is the revision point under consideration. BANNER adds a new node to the parent set of node G in the original network, as shown in Figure 5.7. The heuristic for choosing the new parent is discussed in the next section.

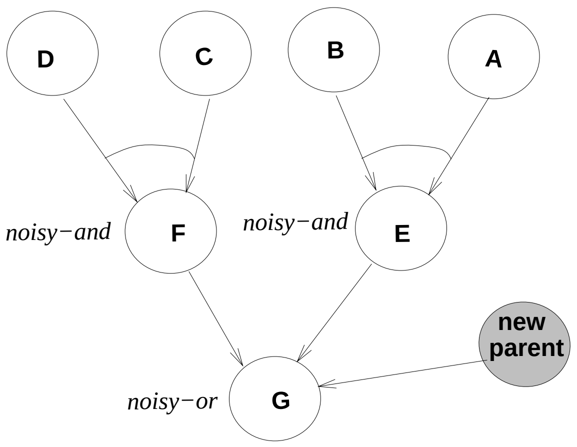


Figure 5.7: Revision operator: Adding a new parent

2. **Adding a new hidden node:** When a candidate revision point is a *link-leak* node, BANNER modifies the link associated with that node. One of the ways that BANNER revises a link is by introducing a new hidden variable between the two nodes of the link, and using the heuristic described in the next section to find a new node to be added to the parent set of the newly introduced hidden node. For example, when the revision point under consideration is the *link-leak* node, *E-A-leak*, from the augmented network shown in figure 5.4, BANNER revises the original network by introducing a new hidden node between nodes E and A, which is of the same type as the intervening node, *E-A-e*, used to augment the link between E and A in the augmented network (Figure 5.8). The parents of the new hidden node are nodes A and a new parent picked by BANNER using the criteria discussed in the next section.

3. **Deleting a link:** When the candidate revision point is a *link-leak* node, BANNER can also decide to delete the link corresponding to the *link-leak* node. Consider again the example where the *link-leak* node *E-A-leak* of the augmented network is the candidate revision point.

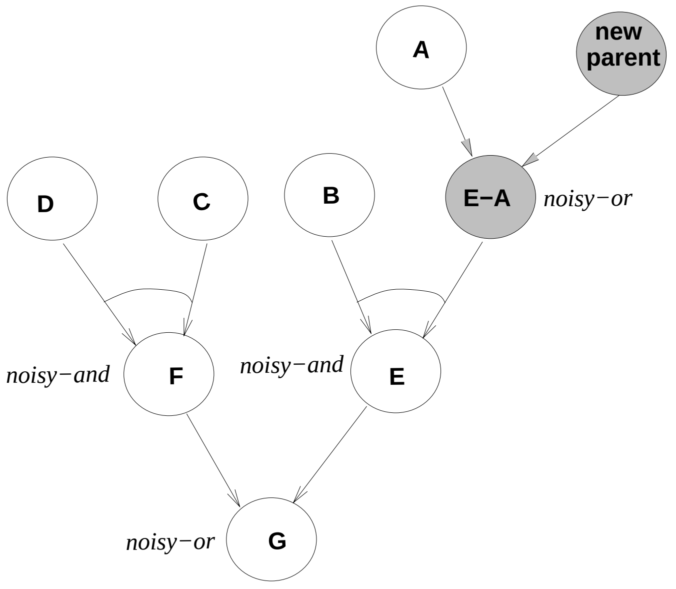


Figure 5.8: Revision operator: Adding a hidden node

Suppose that it is observed that node *E-A-leak* does not need to be false for any of the examples, but needs to be true for some examples. This means that the link between nodes E and A in the original network may be deleted in order to correct the misclassified examples without affecting any of the other examples. An analogous argument can be made for Noisy-Or nodes. This is only one of the conditions under which BANNER deletes a link.

A second condition for deleting for a link arises out of the case where BANNER considers adding a hidden variable as described above. Let us consider the augmented network shown in figure 5.4, where the candidate revision point is the node *E-A-leak*. Suppose that BANNER picks the negation of A to be the new parent of the intervening node E-A (Figure 5.8). Such an addition is equivalent to deleting the link between E and A, and hence, BANNER deletes the link between E and A instead of adding a hidden node as described above.

5.4 Choosing a Parent to Add

The first two revision operators discussed above require the selection of a new parent to be added to a node, which would act as an inhibitor in some cases, and as an enabler in others. Here, we will describe the procedure for selecting a new enabling parent. The procedure for selecting a new inhibitory parent is identical, except that the *information gain* metric, discussed below, would have to be slightly modified. We will point out the necessary changes to this metric once we have presented its details.

The problem of finding a node to be added to the parent set of another node may be viewed as that of finding a feature that best discriminates between two classes, given a set of examples. As mentioned earlier, in addition to a list of misclassified examples which need the enabling influence, BANNER also maintains a list examples (not necessarily all misclassified), for which the new parent should be not be enabled. Thus, there are two sets of examples, one for which the new parent should be true, and one for which the new parent should be false. BANNER uses the *information gain (Gain)* (Mitchell, 1997; Quinlan, 1993, 1990, 1986; Mingers, 1989) metric to choose a parent, from among a set of candidates, that best reflects this distribution.

The *Gain* metric, commonly used in inductive learning algorithms (Mahoney & Mooney, 1994; Quinlan, 1990, 1986), estimates the information gained about a target function value from knowing the value of an attribute. For example, given a single-class problem, a set of binary-valued attributes, and a set of examples consisting of attribute vectors labeled with class membership, the *Gain* metric can be used to pick a feature that best predicts the classification of the examples. There are two versions of this metric that are commonly used. One version, used in decision tree learning algorithms such as ID3 and C4.5 (Quinlan, 1986, 1993), is used in situations where it is sufficient to select a feature that best discriminates between sets of examples, with no constraints on what the value of the feature should be for each set of examples. A second version, used by Quinlan (1990) to learn propositional Horn-clause theories, is used in situations where it is not only necessary to pick a feature that best discriminates between sets of examples, it is also required that the feature have specific values (e.g. true or false) for each set of examples. In our case, we need to select a new parent that discriminates between the examples that need an enabling influence, and the examples that need an inhibitory influence, with the additional constraint that the new parent be true for the former set of examples and false for the latter set of examples. Therefore, we have used the second version of the information gain metric, as described below.

Suppose that we are given a set of examples, S, of size N, of which N^+ are *positive* examples of a given class C, and N^- are *negative* examples of C. Also assume that all the features in the examples are boolean-valued. Then, according to information theory, the number of bits needed to encode the classification of a positive member of the class is given by:

$$I(S) = -\log_2 \left(\frac{N^+}{N^- + N^+} \right)$$

For any given feature F, let N_f be the number of examples for which F is true; of these let, N_f^+ be the number of examples which are positive examples of C, and N_f^- be the number of examples which are negative examples of C. Then, the number of bits required to encode the set of examples for which both F and C are true is given by:

$$I(N_f) = -\log_2 \left(\frac{N_f^+}{N_f^- + N_f^+} \right)$$

Thus, the reduction due to F in the total number of bits required to encode the positive members of C is given by

$$Gain(C, F) = N_f^+ * (I(S) - I(N_f))$$

The higher the value of this function, the greater the correlation between the examples for which F is true and the positive examples of C. Note that this computation can be easily generalized to hidden variables and variables with missing values. Information gain for such nodes can be obtained by weighting the frequency measures N_f^+ and N_f^- by the degree of belief associated with these nodes for each example.

So far, we have described this metric with a view to selecting an enabling parent. The same metric is used to select an inhibitory parent, except that in this case, N_f is defined as the number of examples for which F is false. Every other term in the computation of the metric is defined as before.

Figure 5.9 shows how BANNER uses the information gain metric to select an enabling parent. The inputs to this algorithm are: the node N whose parent set is being augmented, a set of nodes, CF that are candidates for being selected as a new parent, and E, the union of the set of examples for which the new parent should be true and the set of examples for which it should be false. Note that the algorithm assumes that CF does not contain any parents or descendents of node N, since it is unproductive to consider an existing parent to be added as a new parent, and adding a link to the node from one of its descendents would lead to a directed cycle. The algorithm returns an enabling node, unless none of the nodes in CF leads to positive information gain. The algorithm for selecting an inhibitory parent is identical, except that the *Gain* metric is modified as described above.

```
get-enabler($N, CF, E$)
    1. For each $F \in CF$, compute $Gain(E, F)$
    2. Set $F_{max} = argmax_{f \in CF}\ Gain(E, f)$.
    3. If $Gain(E, F_{max}) \leq 0$,
        (a) Return $NO\text{-}ENABLER\text{-}FOUND$
    4. Else
        (a) return $F_{max}$
```

Figure 5.9: Algorithm for Selecting a New Enabling Parent

In the most general case, all the nodes in the network, observed and hidden, and their negations are candidates for the new parent. However, since adding links from hidden variables may violate the constraint imposed by BANNER-PR that the network be a *virtual polytree*, we restrict the set of candidate parents to just the observed variables for problems where BANNER-PR is used as the parameter revision module. In addition, we do not consider negations of those variables that are multi-valued variables converted into binary-valued variables to fit the representation. Figure 5.10 shows the summary of the theory revision algorithm.

5.5 Stopping Criteria

Deciding when to stop training, so that the network learns the training data effectively without overfitting, is crucial. Each of the revision components, namely the parameter revision and the structure revision components, could lead to overfitting. Section 4.1.3 discusses the details of our technique for deciding when to stop parameter revision. BANNER uses information from the parameter revision module as a part of its criteria for deciding when to stop revising the network. Thus, BANNER stops training when the parameter revision module reports overfitting (Section 4.1.3 or reports one hundred percent accuracy on the training set. In addition, BANNER also stops training when it observes that the training accuracy of the network has not improved for a certain

Given: An initial network, and a set of training data.
Output: A revised network.
Algorithm:

1. Initialize the parameters of the network either randomly or based on some prior knowledge.
2. Repeat steps a-e until further training does not improve accuracy on the training data.
   (a) set *train-net* = initial network.
   (b) set *leak-net* = train-net augmented with *node-leak* nodes.
   (c) Train network *train-net* to revise parameters.
   (d) If the previous step indicates overfitting, *or* all examples are correctly classified, return *train-net.*
   (e) else
      i. Train network *leak-net* to estimate prior probabilities of the *node-leak* nodes.
      ii. Set *augmented-net* = *train-net* augmented with *node-leak* and *link-leak* nodes.
      iii. Copy priors of leak nodes from *leak-net* to *augmented-net.*
      iv. For each example,
         A. Instantiate input and target nodes of *augmented-net* with values from the example.
         B. Infer beliefs of all the nodes in *augmented-net.*
         C. Collect all enabled and inhibited *node-leak* and *link-leak* nodes.
      v. Set *revision-points* =small set of *node-leak* and *link-leak* nodes that cover all the misclassified examples found using the greedy set cover algorithm.
      vi. For each revision point in *revision-points*, revise *train-net* at the revision point using one of the revision operators.

Figure 5.10: Detailed outline of the algorithm used by BANNER

number of iterations.[2]

5.6 Variants of the Algorithm

The previous sections described the theory refinement algorithm used by BANNER. In this section, we describe some variations of the algorithm that may be useful in some learning scenarios.

5.6.1 As an Inductive Learner

BANNER has been designed for the purpose of using examples to revise a given theory using examples, in order to improve its classification accuracy on those examples. However, BANNER can learn networks even when such a theory is not available, by constructing a default network, and performing theory refinement using this as the initial network. There are several ways to construct an initial default network. One is to build a two-layered network by connecting each target variable to each input feature. Figure 5.11 shows one such network where variables A, B, C, D, G and E are the variables observed in the data. Here, nodes G and E could either be Noisy-Or or Noisy-And nodes. Given this network, BANNER would have to prune away unnecessary links and introduce hidden variables if necessary. Another possibility is to construct a network that is completely disconnected, i.e. none of the nodes in the network are connected. In this case it would be necessary to provide some ordering on the variables by specifying some to be the sources of the network, and some to be the sinks. In our experiments, we decided to use the latter approach, since the former approach results in very large initial networks. We will discuss the effectiveness of BANNER for inductive learning in the next chapter.

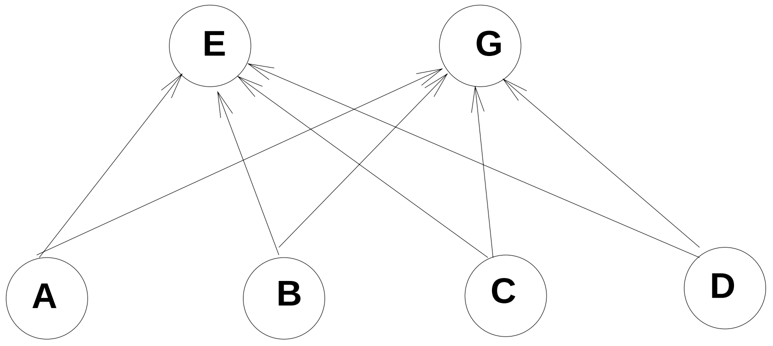


Figure 5.11: An initial network for inductive learning

5.6.2 Learning Networks with Leak Nodes

The previous sections described how BANNER uses *leak* nodes to localize portions of the network to be revised. However, it may be desirable to learn networks with leak nodes when it is suspected or known that the data may not contain all the information necessary to model the domain. An example of this would be a scenario where the goal is to learn a network that models data describing set of symptoms and set a diseases, but it is known that not all causes of the symptoms

[2]This number was set to 3 for all our experiments.

are represented in the data. In fact, learning networks with leak nodes in such situations provides another way of introducing hidden variables into the network. BANNER can be used in these situations with only a slight modification to the algorithm. Since now the goal is to learn networks that include leak nodes, the two parameter revision steps in each iteration (steps 2c and 2ei in Figure 5.5), namely one to learn the parameters and one to estimate the prior probabilities of the network, can be replaced by one parameter revision step that estimates the parameters of the network that includes leak nodes. The rest of the algorithm remains unchanged. Section 6.6 describes an experiment in which we used *Banner* to learn a network with leak nodes and discusses its performance.

Chapter 6

Empirical Evaluation

In this chapter, we will to show through experiments on realistic problems and data, that our technique is effective in revising networks to significantly improve their accuracies on a given classification task. We will also compare the performance of BANNER with some other learning algorithms. Finally, we will study the contribution of the different components of BANNER to its overall effectiveness.

One way to determine if our theory refinement algorithm achieves its purpose is to use it to revise some theories that are known to be approximate and measure the accuracy of the revised Bayesian network on independent test data. An improvement in accuracy would provide evidence that the algorithm has indeed achieved its purpose.

While it is important to study an algorithm's effectiveness in achieving its goal, it is also important to evaluate its performance against other algorithms designed for the same purpose. BANNER is a theory refinement algorithm based on the hypothesis that it is more effective to learn from an initial approximate theory than to learn from scratch. In order to test this hypothesis, we will compare the performance of BANNER with some inductive learning algorithms such as C4.5 (Quinlan, 1986), which is a well-known technique for learning decision trees, the Naive Bayes algorithm, BACKPROP, which is the neural network backpropagation algorithm described in Section 2.2, as well as an inductive version of BANNER. The Naive Bayes algorithm learns a special class of Bayesian net classifiers (Section 2.4), and has been shown to produce highly accurate classifiers, and in many cases, is hard to beat. Therefore, comparisons with the Naive Bayes algorithm will also serve to illustrate the usefulness of BANNER specifically as a tool for learning Bayesian network classifiers.

BANNER can also be used as a hybrid learning algorithm that combines symbolic and probabilistic representations. Therefore, we compare its performance with other hybrid learning algorithms such as KBANN (Towell & Shavlik, 1994) and RAPTURE (Mahoney, 1996; Mahoney & Mooney, 1993), and with EITHER (Ourston & Mooney, 1994) which is a technique for revising logical theories. Since BANNER, unlike other hybrid learning systems, is capable of revising theories even when the intended prediction task is diagnostic as opposed to causal, we will evaluate it performance on one such problem.

Finally, not only are we interested in overall performance of BANNER, we would like to evaluate the contribution of each individual component to the overall algorithm. In particular, we would like to study the extent to which structure revision contributes to the overall performance of Banner, and the benefits of starting out with an initial theory. For this, we perform *ablation* studies, where we disable certain components of the algorithm, and compare the performance of the ablated system with that of the full system. We will study the following ablated versions of our system: Banner-Ind, an inductive version of Banner which incorporates both the parameter revision and structure revision component but uses a default initial theory, as discussed in Section 5.6.1 , and Banner-Pr (parameter revision), which uses a given initial theory but does not perform any structure revision. We will compare the performances of these ablated systems with that of Banner on each of the experimental domains.

The standard practice in evaluating algorithms that learn Bayesian networks is to assume that the target network, commonly called the *gold standard*, is known ahead of time (Cooper & Herskovits, 1992; Aliferis & Cooper, 1994; Russell et al., 1995). Given the target network, the practice is to generate data from the network, learn a Bayesian network from the generated data, and compare the learned network with the original. Such evaluations assess the ability of an algorithm to learn a Bayesian network from data, when it is known that there is a Bayesian network that models the data. In real-world learning situations, the target concept is not known in advance, nor is it known what the most suitable representation for the target concept would be. The success of a technique on real-world learning problems is determined not only by its ability to learn effectively, but also on the suitability of the representation, and its underlying assumptions, for the domain. While experiments with artificial data demonstrate the effectiveness of the learning algorithm, they do not say anything about the usefulness of the representation for modeling real domains. By evaluating techniques on real data sets, we not only demonstrate the effectiveness of the learning algorithm, but also provide evidence that the underlying representation and assumptions are realistic in that they model the real world effectively. With this in mind, we we will study the performance of Banner on some real-world data sets for which we not know the target theories, but do have some initial approximations of the theories. These data sets also demonstrate that our technique can be effectively applied to fairly complex domains involving hundreds of variables.

In the following sections, we will evaluate Banner on five classification problems: recognizing DNA promoters (Noordewier et al., 1991), recognizing DNA splice-junctions (Noordewier et al., 1991), learning student models for a C++ tutor (Baffes, 1994), diagnosing brain disorders in human patients (Tuhrim et al., 1991), and classifying chess end-games (Shapiro, 1983, 1987). The first four of these problems have associated domain theories, represented as propositional Horn-clause rules, that do not have good predictive accuracies on the data and therefore need to be revised. The fifth problem does not have an initial domain theory and is intended to study the effectiveness of Banner in learning networks from scratch. We will also evaluate the structure revision component of Banner more directly by performing experiments on corrupted versions of the theory for recognizing DNA promoters. The data sets and domain theories for all these problems, except the student modeling problem, can be obtained from the University of California, Irvine repository of machine learning databases (Merz, Murphy, & Aha, 1996).

The results reported here for all techniques, other than Banner and the Naive Bayes algorithm, were obtained from other sources. The results for Rapture, Either, *C4.5*, and Backprop were obtained from Mahoney (1996), and those for Kbann were obtained from Towell (1991). The results on the C++ tutoring domain, for systems other than Banner, Banner-Ind, and Naive

BAYES, were obtained from Baffes (1994). The results reported for the Naive Bayes algorithm were generated using an implementation of the algorithm that uses the Laplace estimation strategy, as discussed in section 2.4, to avoid the problem of zero estimates. Henceforth, we will refer to this particular implementation as NAIVE BAYES.

6.1 Experimental Methodology

The purpose of learning is to acquire knowledge that can be applied to novel situations. Thus, all learning techniques should be evaluated and compared on the basis of their ability to generalize to new situations. This means that techniques that learn from examples should be evaluated on how well they can generalize to examples that are not among the ones used to train the system. A standard methodology for evaluating such a learning technique, is to partition the given set of examples into a *training set* and a *test* set, present the learning algorithm with examples from the training set, and measure the *accuracy* of the learned theory on the test set, where accuracy is defined as the percentage of examples that are classify correctly.

It is also common to evaluate learning algorithms on training sets of different sizes, and generate *learning curves* that chart the performance of the algorithm on training sets of increasing sizes. The slope of a learning curve is an indicator of how much performance can be gained by increasing the number of training examples. Learning curves also provide an insight into the limit beyond which increasing the number of training examples will not result in significantly improved performance. Studying the performance of different learning algorithms on data sets of varying sizes helps in the selection of a learning algorithm that will lead to the best performance for the amount of data available.

While comparing learning algorithms, it important to not judge their performances on a single trial, but rather on their performances averaged over many trials on different *training* and *test* sets. There are several methodologies for doing this. The *re-sampling* methodology conducts a series of about 20 to 30 trials, where during each trial, the given data set is randomly partitioned into a training set and a test set, which are then used to evaluate each learning algorithm, and the results are averaged over the trials. The *k-fold cross-validation* methodology randomly partitions the given data set into k sets, $T_1, \cdots, T_k$. The results are averaged over k trials, where each trial i uses T_i as the test set, and the union of all the remaining sets $T_j, j \neq i$, as the training set. The latter approach has been shown to be more indicative of the true differences between performances of learning algorithms than the former (Dietterich, 1998). However, the former methodology has been very commonly used in the literature, and many of the learning techniques such as EITHER, RAPTURE, and KBANN have been evaluated using this methodology. We use the re-sampling methodology whenever the training/test splits used to evaluate these techniques were available to us, so as to compare their performances with that of BANNER when trained on the same set of training examples. Otherwise, we use the *cross-validation* methodology.

The learning curves for RAPTURE, KBANN, EITHER, C4.5 and BACKPROP were generated using the *re-sampling* methodology, where during each trial, starting from an empty set, the size of the training set was increased successively, while the test set was held constant. As mentioned earlier, we used the *re-sampling* methodology to evaluate BANNER whenever the training/test set splits were available. However, instead of holding the test set constant during each trial, while adding to the training set, we used the approach of including in the test set, all the data not included in the training set. This means that the size of the test set decreased as the size of the training set increased. Thus, while all the systems were trained on identical training sets, BANNER and NAIVE BAYES were evaluated on test sets not identical to those used to test the other systems. This mainly affects the choice of the procedure to be used to test the statistical significance of the differences in performances between these systems. Tests for statistical significance are necessary to determine whether the differences in the results obtained for various techniques are merely due to statistical variations, or due to the inherent differences between the techniques. It is necessary to the use the *unpaired t-test* (Siegel, 1988) to test for statistical significance of the differences in the results obtained for systems that were not evaluated on identical test sets, since the results for each trial are no longer completely paired. The significance of the differences in the results obtained for BANNER and NAIVE BAYES, which were evaluated on identical test data, can be tested with *paired t-test* (Siegel, 1988). The unpaired t-test is the more conservative test of the two.

6.2 DNA Promoter Recognition

DNA is the basis of genetic information in all living creatures. A DNA molecule consists of two strands of nucleotides, where some sequences of nucleotides contain information for the synthesis of important proteins, and are called *genes.* Given a strand of DNA, geneticists are interested in identifying those sequences of nucleotides that form a gene. This is complicated by the fact that gene sequences are separated by sequences of nucleotides that do not encode any useful information. Thus, it is important to have techniques for identifying those portions of the DNA strand that encode genetic material. It has been observed that gene sequences are always immediately preceded by particular sequences of nucleotides called *promoters*, so that identifying promoter sequences would help identify the start of gene sequences. This can be viewed as a classification problem, where the task is to classify a sequence of nucleotides according to whether or not it is a promoters. Since geneticists have not yet formulated the rules for classifying promoters accurately, there has been significant interest in learning such classifiers from data. Based on an analysis of biological literature found in O'Neill and Chiafari (1989), Noordewier et al. (1991) have developed a set of rules for recognizing promoters. However, this theory is overly specific, which makes it an ideal candidate for theory refinement.

There are two sets of data associated with this domain. The first set, consisting of 106 examples (53 positive examples, and 53 negative examples), has been widely used to evaluate several inductive and theory refinement systems. The second data set has 468 examples, 234 of which are positive examples of the class. Each example has 57 input features, which represent nucleotides, each of which can take on one of four values, A, G, T or C. Each example is also labeled according to whether or not it is a promoter. The input features represent a window over a DNA strand. The information in this window is used to determine the presence of a promoter. Figure 6.1 (Mahoney, 1996) shows a sequence of nucleotides that form the input features. The nucleotide labeled $P1$ marks the beginning of a potential gene sequence. The task is to determine whether the nucleotides labeled P-50 to P-1, immediately preceding the potential gene, form a promoter.

Figure 6.2 shows the initial logical theory for recognizing a DNA promoter sequence. Fig-

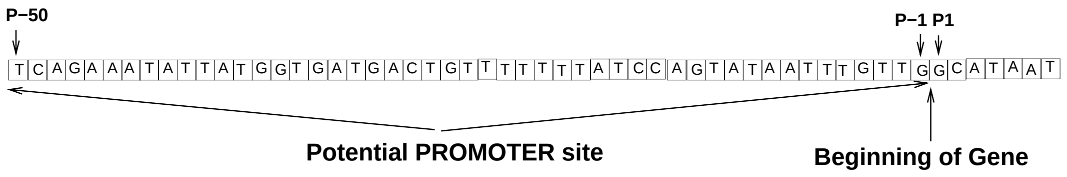

P−50
P−1 P1
T C A G A A A T A T T A T G G T G A T G A C T G T T T T T T T A T C C A G T A T A A T T T G T T G G C A T A A T
Potential PROMOTER site
Beginning of Gene

Figure 6.1: Example of a DNA string

ure 6.3 shows a portion of the Bayesian network derived from this theory, as discussed in Section 3.3. We will first discuss experiments with the data set with 106 examples, followed by a discussion of the experiments with the larger data set.

promoter	⟵	contact, conformation
contact	⟵	minus_35, minus_10
minus_35	⟵	(p-37 c) (p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)
minus_35	⟵	(p-36 t) (p-35 t) (p-34 g) (p-32 c) (p-31 a)
minus_35	⟵	(p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c) (p-31 a)
minus_35	⟵	(p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)
minus_10	⟵	(p-14 t) (p-13 a) (p-12 t) (p-11 a) (p-10 a) (p-9 t)
minus_10	⟵	(p-13 t) (p-12 a) (p-10 a) (p-8 t)
minus_10	⟵	(p-13 t) (p-12 a) (p-11 t) (p-10 a) (p-9 a) (p-8 t)
minus_10	⟵	(p-12 t) (p-11 a) (p-7 t)
conformation	⟵	(p-47 c) (p-46 a) (p-45 a) (p-43 t) (p-42 t) (p-40 a) (p-39 c) (p-22 g) (p-18 t) (p-16 c) (p-8 g) (p-7 c) (p-6 g) (p-5 c) (p-4 c) (p-2 c) (p-1 c)
conformation	⟵	(p-45 a) (p-44 a) (p-41 a)
conformation	⟵	(p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t) (p-16 t) (p-15 g) (p-1 a)
conformation	⟵	(p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t) (p-21 a) (p-20 a) (p-17 t) (p-15 t) (p-4 t)

Figure 6.2: DNA Promoter Recognition - Initial Domain Theory

6.2.1 Experiments with the 106-example Data Set

This data set consists of 106 examples, of which 53 are positive and 53 are negative. None of the examples has any missing values for any of the input features. Since this problem satisfies all the constraints on the applicability of Banner-Pr, namely that the data has no missing values, the evidence renders the initial network to be a virtual polytree, and the classification task involves causal prediction, these experiments were performed using Banner-Pr for parameter revision.

The learning curves for Banner and Naive Bayes were generated using the *re-sampling* methodology involving 25 trials with training sets of increasing size. The results reported for the other systems, excluding Kbann, were obtained from Mahoney (1996), and were also generated using the re-sampling methodology over 25 trials, each using the same training sets as the experiments with Banner and Naive Bayes. The results for Kbann were obtained from Towell (1991), and were generated using training sets different from those used with the rest of the systems. Figure 6.4

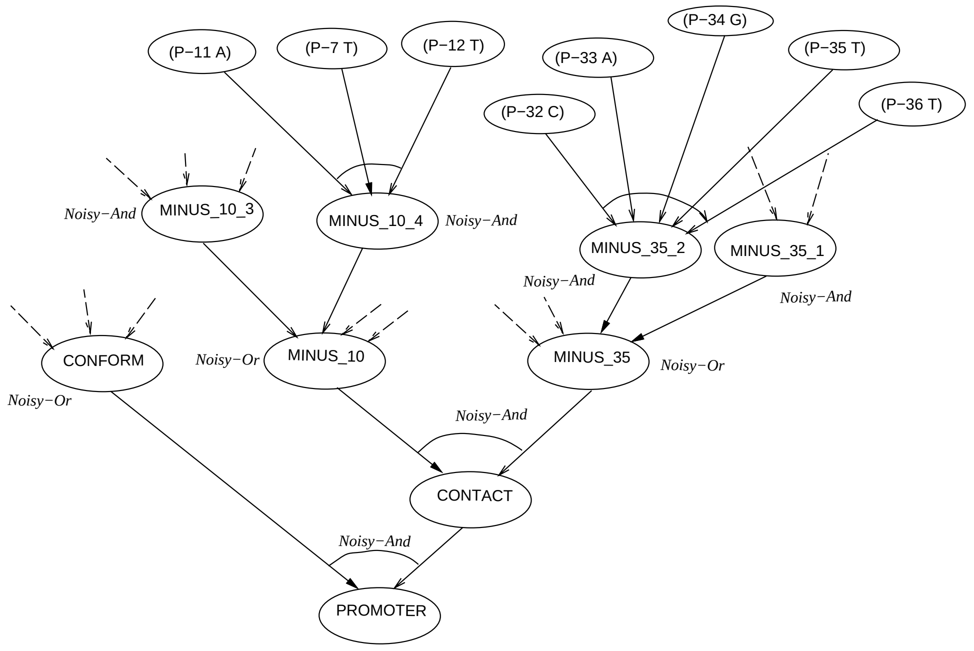


Figure 6.3: DNA Promoter Recognition - Bayesian Network

shows the learning curves for these techniques on this problem. Figure 6.5 shows the performances of the ablated versions of BANNER.

Before any revision takes place, the network derived from the initial theory has an accuracy of 50%. The learning curve for BANNER clearly demonstrates that it is successful in improving the accuracy of the initial theory substantially (by about 40 percentage points) with only 40 examples. While its learning curve is not as steep as those for RAPTURE or KBANN, BANNER catches up with these systems at 40 examples, and performs comparably thereafter. The differences between the learning curves of BANNER and RAPTURE are statistically significant at the 0.02 level or less for 10 and 20 examples, and are not significant for the rest of the points on the learning curve. In contrast, BANNER performs better than the theory refinement algorithm, EITHER, by large margins throughout. This is not surprising because this domain is known to require evidence-summing, a facility that logical theories lack, in order to perform well. We were unable to test the statistical significance of the differences between the learning curves of BANNER, KBANN, and EITHER since we did not have the results from each of the trials for the latter systems.

The advantage of leveraging off of an initial theory is brought out by the fact BANNER outperforms the inductive learning systems, NAIVE BAYES, BACKPROP and C4.5 by large margins, on every point in the learning curve. This is also illustrated in Figure 6.5, where the learning curve of BANNER is steeper, and higher than that of BANNER-IND, the inductive version of BANNER. BANNER-IND performs comparably with NAIVE BAYES (the differences in performances are not statistically significant), which lends evidence to its effectiveness as an inductive learner. The difference in performance between BANNER and NAIVE BAYES is statistically significant for all points in the learning curve at the 0.05 level or less, and the difference in performance between BANNER and BANNER-IND is statistically significant at all points in the learning curve, except at 90 examples, at the 0.05 level or less.

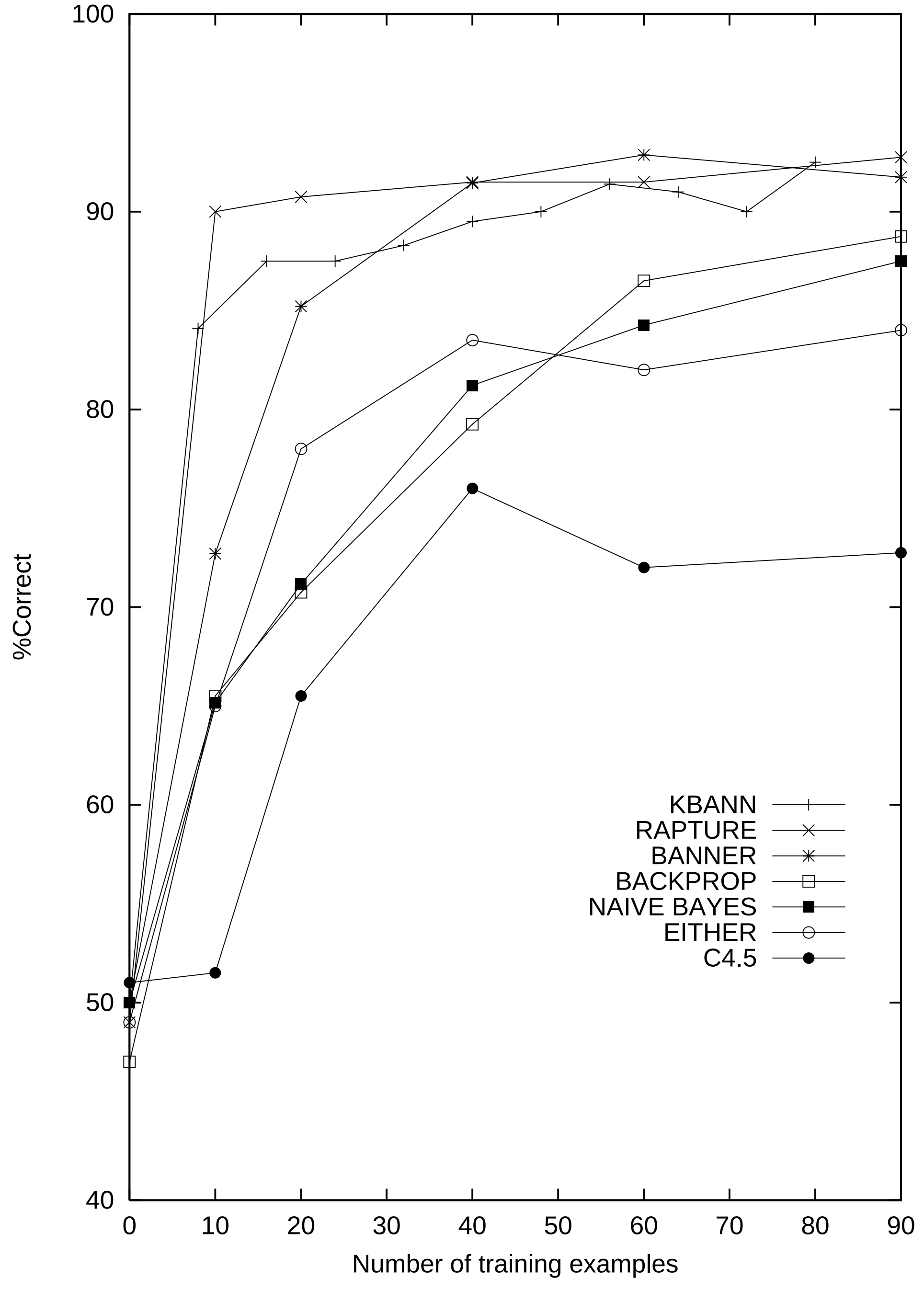


Figure 6.4: DNA Promoter Recognition: Performance of various systems on the smaller data set

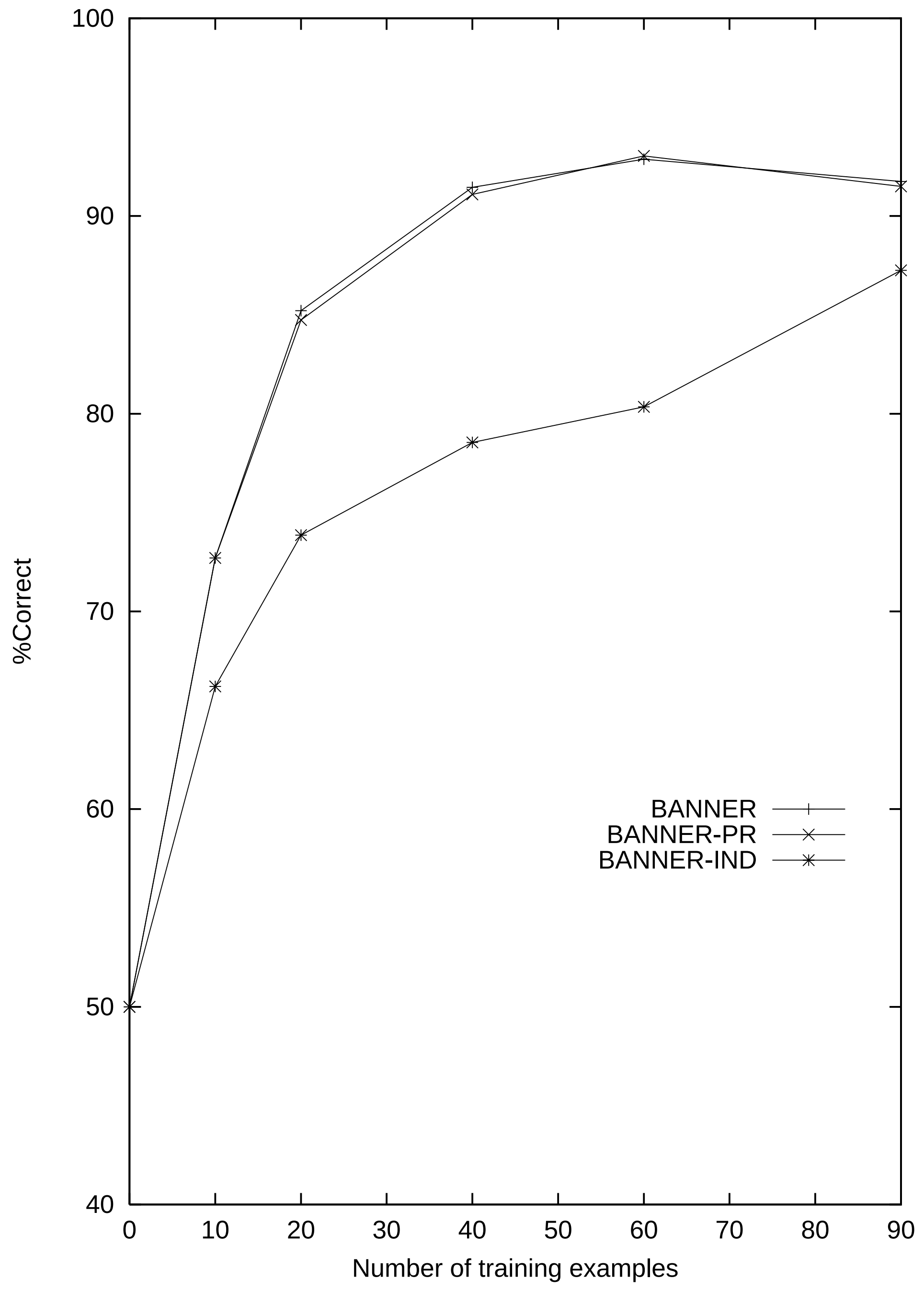


Figure 6.5: DNA Promoter Recognition: Banner ablations on the smaller data set

Finally, the differences in performances of BANNER and BANNER-PR are not statistically significant, showing that just revising the parameters is sufficient to produce a highly accurate classifier for this problem. In fact, the structure revision component was never invoked for this data set, since parameter revision alone was sufficient to fit the data. The observation that just enhancing the domain theory with appropriate numeric parameters, without any modifications to the structure of the theory, is sufficient to produce highly accurate classifiers for this domain is corroborated by previous studies of this domain (Mahoney, 1996; Koppel et al., 1994).

6.2.2 Experiments with the 468-example Data Set

This larger set of examples comes from the Human Genome Project (Alberts, 1988). This data set consists of 234 positive examples and 234 negative examples. The main difference between this and the smaller data set is that about 15% of the examples have missing values for one or more of the input features and violate the conditions, described in Section 3.2, under which BANNER-PR may be used to learn from data with missing observations. Specifically, some of the examples are missing values for evidence variables that are required to break loops. However, as discussed in Section 4.3, the long training time required by C-APN on this domain precludes its use for parameter revision. For this reason, and because the number of examples that violate the assumptions of BANNER-PR is very small, the following experiments used BANNER-PR to revise the parameters, discarding the inappropriate examples from the training set. However, none of the examples in the tests sets used to evaluate the final networks were discarded.

These experiments were performed using the *re-sampling* methodology with 20 trials, with the same training splits used by Mahoney (1996) to evaluate RAPTURE, BACKPROP and C4.5. The results reported for these systems were obtained from Mahoney (1996). The results for KBANN were obtained from Towell (1991), and were generated using a different set of training/test splits.

Figures 6.6 shows the learning curves for the different learning algorithms on this problem, and Figure 6.7 shows the performances of the ablated versions of BANNER. The results on this data set are similar to those obtained with the smaller data set, discussed in the previous subsection. Before any revision, the accuracy of the network derived from the initial theory is only 50%. The learning curve for BANNER clearly demonstrates that it is highly effective in improving the accuracy of the initial theory, with improvements of as much as 38% percentage points obtained with only 50 training examples.

The decision tree learning algorithm, C4.5, performs substantially worse than the rest of the techniques, all which learn representations with numeric parameters. The graph shows that BANNER starts out performing significantly worse (at the 0.01 level) than RAPTURE on training sets with 50 examples or less, catches up with it at 200 examples, and performs significantly better (at the 0.01 level) on training sets with 400 examples. KBANN performs slightly better than BANNER on training sets with 300 examples and less, although we were unable to test the statistical significance of these differences because we do not have access of the results obtained with KBANN on each individual trail. The theory refinement techniques perform considerably better than the inductive learners, NAIVE BAYES, BACKPROP, and BANNER-IND on smaller training sets, once again illustrating the advantage of starting with an initial, approximately correct theory. The learning curve of BANNER is considerably higher than that of NAIVE BAYES for training sets with less than 100 examples and for training sets with 400 examples (significant at the 0.01 level), while it is significantly lower for 200 examples (significant at the 0.05) level. BANNER performs substantially better than BACKPROP throughout, although we could not test the statistical significance of these differences.

Similar to the results observed for the smaller data set, both BANNER and BANNER-PR perform almost identically on this data, except with training sets with 200 examples where BANNER-PR performs slightly better. This difference is statistically significant at the 0.05 level, while the differences for the rest of the points on the learning curve are not significant. Although the structure revision component was invoked for some of the trials with larger number of examples, it did not lead to any significant improvement in performance. This is because just revising the parameters resulted in networks with high accuracy on the training set, and the structure revision algorithm was invoked only to fit a few outliers, and therefore did not result in improved generalization. The networks produced by BANNER are significantly and substantially more accurate than those learned by BANNER-IND (significant at the 0.05 level or less for all points in the learning curve), illustrating the contribution of the initial theory to learning accurate networks. The observation that NAIVE BAYES outperforms BANNER-IND significantly on training sets with 50 or more examples (significant at the 0.001 level) is not surprising, since previous studies have shown that the the Naive Bayes algorithm is very effective on this domain, often outperforming competing techniques (Kohavi et al., 1997).

6.3 DNA Splice-Junction

It has been determined that only about 10% of the nucleotides in human DNA encode information useful for protein synthesis. About 90% of the nucleotides do not encode any useful information and are unutilized. During protein synthesis, sequences of unutilized nucleotides are spliced out, and the remaining nucleotides are used to build proteins. The regions of DNA encoding information are called *Exons*, and the unutilized regions are called *Introns*. The junctions between these regions are called *splice-junctions*, of which there are two kinds: IE sites which are at Intron-Exon boundaries, and EI sites which are at Exon-Intron boundaries. The classification task is that of identifying whether a sequence of nucleotides constitutes an IE or IE boundary.

The data set for this domain, consisting of 3190 examples, was gathered by M. Noordewier (Noordewier et al., 1991). Of these 3190 examples, 768 (24%) are classified as examples of IE borders, 767 (24%) are examples of EI borders, and 1655 (52%) examples contain neither IE nor EI borders. Each example is a string of 60 nucleotides, composed of 30 nucleotides on either side of the hypothesized boundary. These nucleotides form the input features and take on the values A, C, G, or T. The domain theory rules for recognizing these sites from patterns of nucleotides around them (Section B.2) were derived by M. Noordewier from information found in Watson, Roberts, Steitz, and Weiner (1987). The initial logical domain theory was converted into a Bayesian network as discussed in Section 3.3.

Mahoney (1996) evaluated RAPTURE, BACKPROP and C4.5 on this problem, using the re-sampling methodology to run 20 trials with different sets of training and test splits. Rather than use the entire set of 3190 examples, he selected a random subset of 900 examples, which were then randomly partitioned into 20 sets of training/test splits, and used to generate the results reported here.

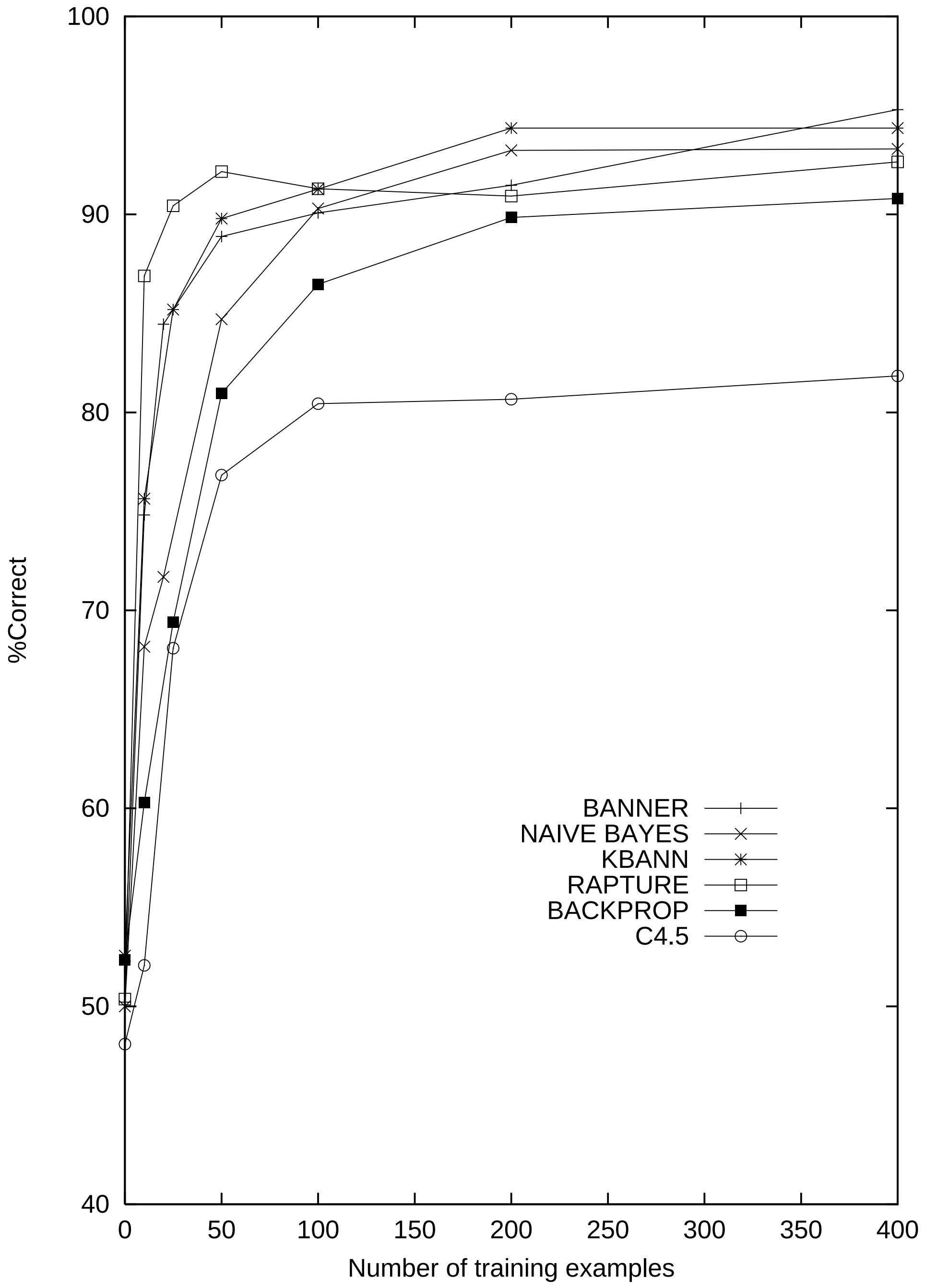


Figure 6.6: DNA Promoter Recognition: Performance of various systems on the larger data set

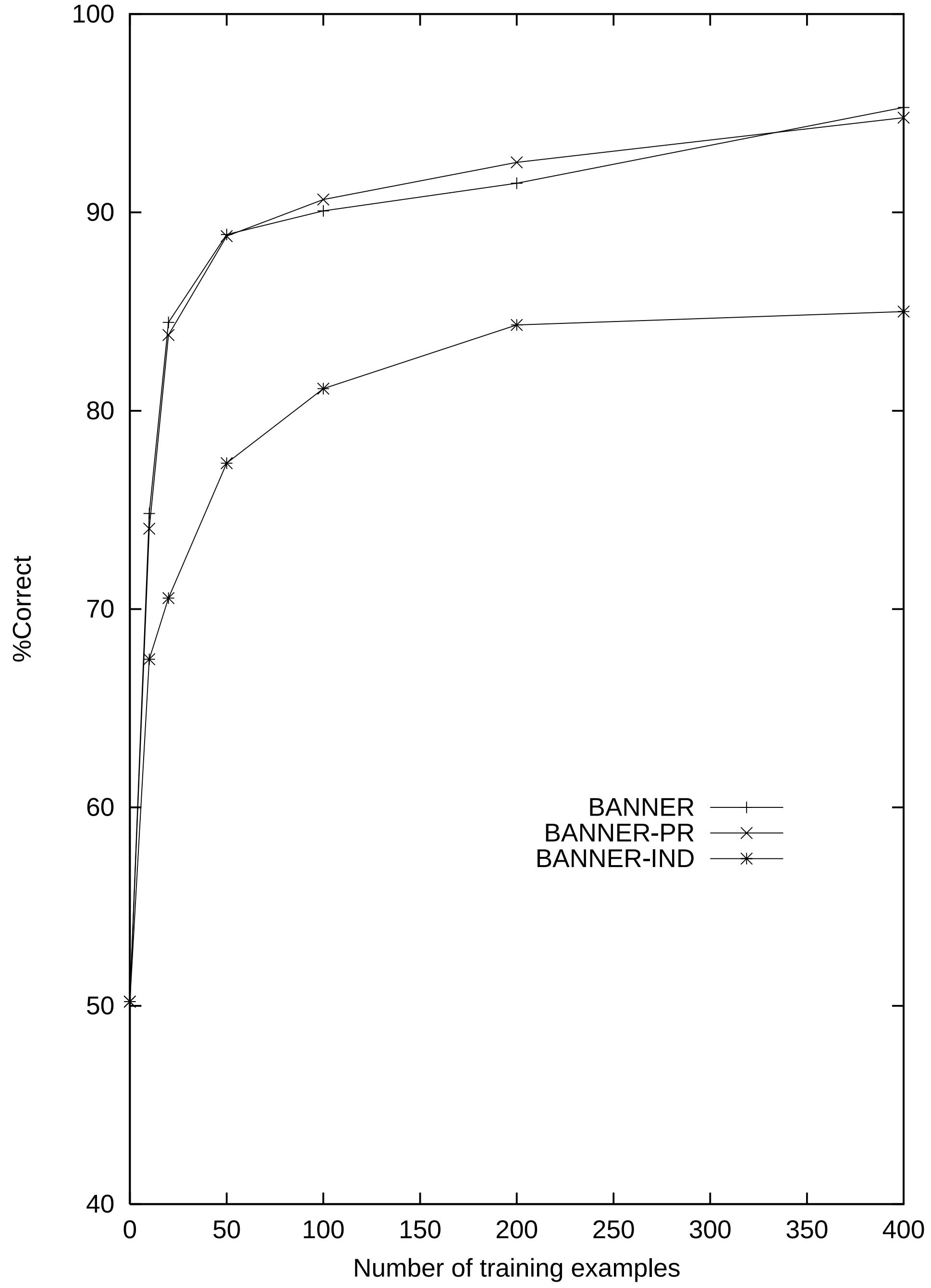


Figure 6.7: DNA Promoter Recognition: Banner ablations on the larger data set

Since the data used in each of the trials used to evaluate the above systems were available to us, we used the same methodology and data to evaluate BANNER and NAIVE BAYES. The results for KBANN were obtained from Towell (1991), and were generated using a different set of training/test splits.

Figures 6.8 shows the performance of these systems on this problem, and Figure 6.9 shows the performance of BANNER ablations. This experiment provides more evidence that BANNER is successful in improving the accuracy of the initial theory significantly with just a small number of examples. The accuracy of the initial theory has risen from 55%, before revision, to 80% when trained on just 20 examples, and to about 94% when trained on 400 examples.

Figure 6.8 shows that the performances of the three hybrid learning algorithms RAPTURE, BANNER, and KBANN are similar, although RAPTURE performs slightly better throughout. The differences between RAPTURE and BANNER are statistically significant for all points in the learning curve at the 0.001 level. As expected, theory refinement algorithms outperform inductive algorithms, BACKPROP, NAIVE BAYES, and C4.5 and BANNER-IND, by a large margins, although NAIVE BAYES catches up with RAPTURE at 200 examples. The differences between the learning curves of BANNER and NAIVE BAYES are significant at the 0.001 level for 20, 50, 100, where the former performs considerably better, at the 0.001 level for 400 examples where it performs slightly worse. We were unable to determine the statistical significance of the differences between BANNER and the rest of the systems for which we did not have the results from each individual trial.

Figure 6.9 demonstrates that the structure revision component of BANNER contributes significantly to its performance on smaller training sets. Structure revision has contributed to an improvement in accuracy of about 14% over BANNER-PR for 20 examples (significant at 0.001 level), and an improvement of about 2.5% for 50 examples (significant at the 0.01 level). The revisions that contributed the most to this improvement were deletions of the links between nodes IE and PR, and nodes EI and $P5G$. As expected, starting out with an initial theory gives BANNER a significant edge over BANNER-IND. The difference in performance between these systems is statistically significant for all points on the learning curves at levels of at least 0.01.

6.4 C++ Tutor

This data set is taken from Baffes (1994), which discusses an *intelligent tutoring System* (ITS) based on the idea of theory refinement. The purpose of this tutoring system is to present students with exercises on some specific topic, and provide explanations of any mistakes they -make. The tutor customizes its explanation to each student based on a model of the student's knowledge of the domain. This *student model* is also used to generate further exercises. The tutor is initially given knowledge about the domain in the form of propositional Horn-clause rules. Given a student's response to a set of questions, the tutor revises the initial domain theory to model of the student's knowledge of the domain. This is an interesting twist on the traditional use of theory refinement, where it is assumed that the data reflects the true domain model, and any discrepancy between the initial domain theory and the data is attributed to a flaw in the theory. In contrast, the tutoring application regards the initial theory as the true model of the domain, and uses the responses from students as data to learn how the student's understanding of the domain differs from the true model. The rationale for such an approach is that many students at least understand the domain

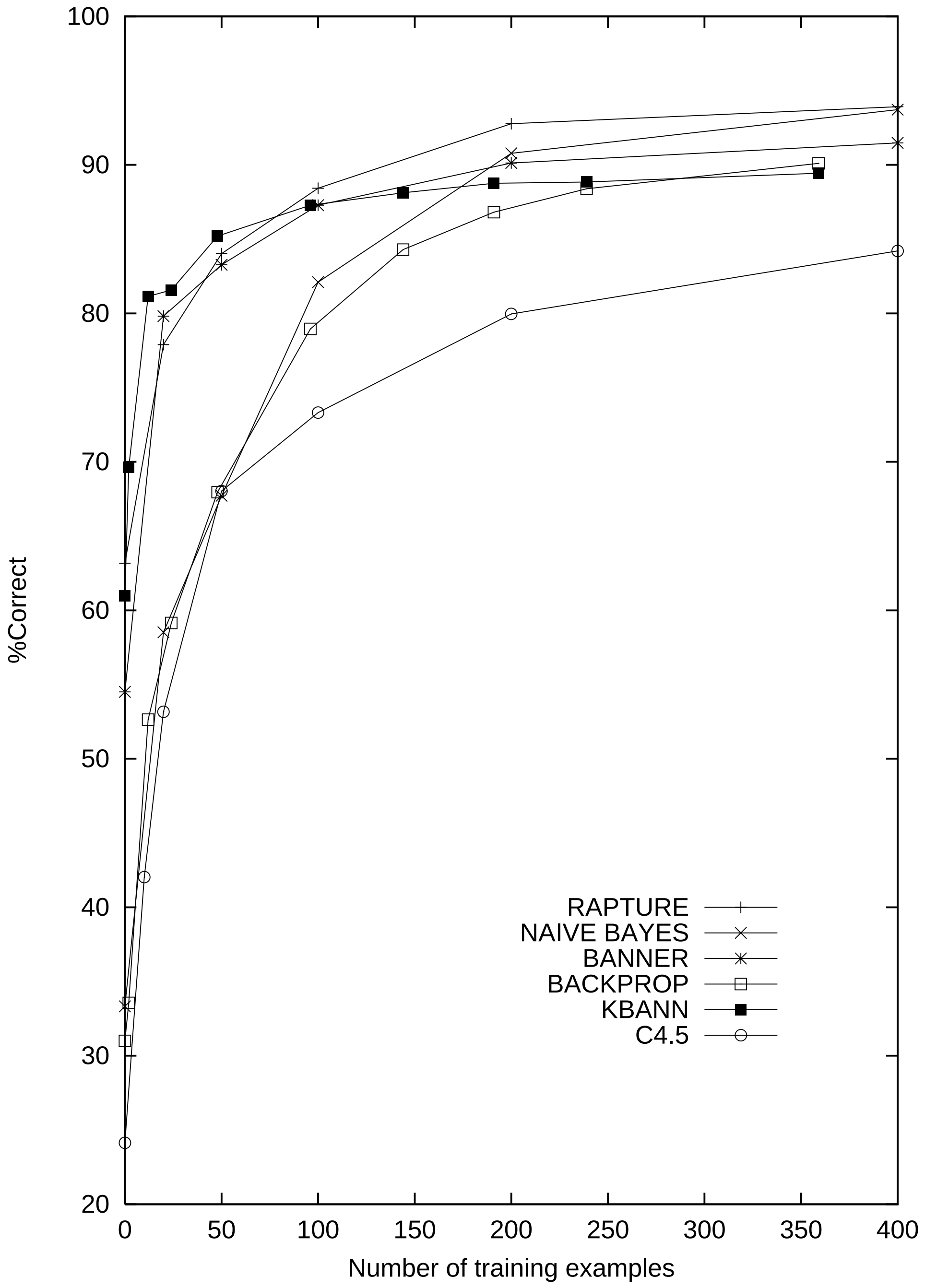


Figure 6.8: Splice-Junction: Performance of Various Systems

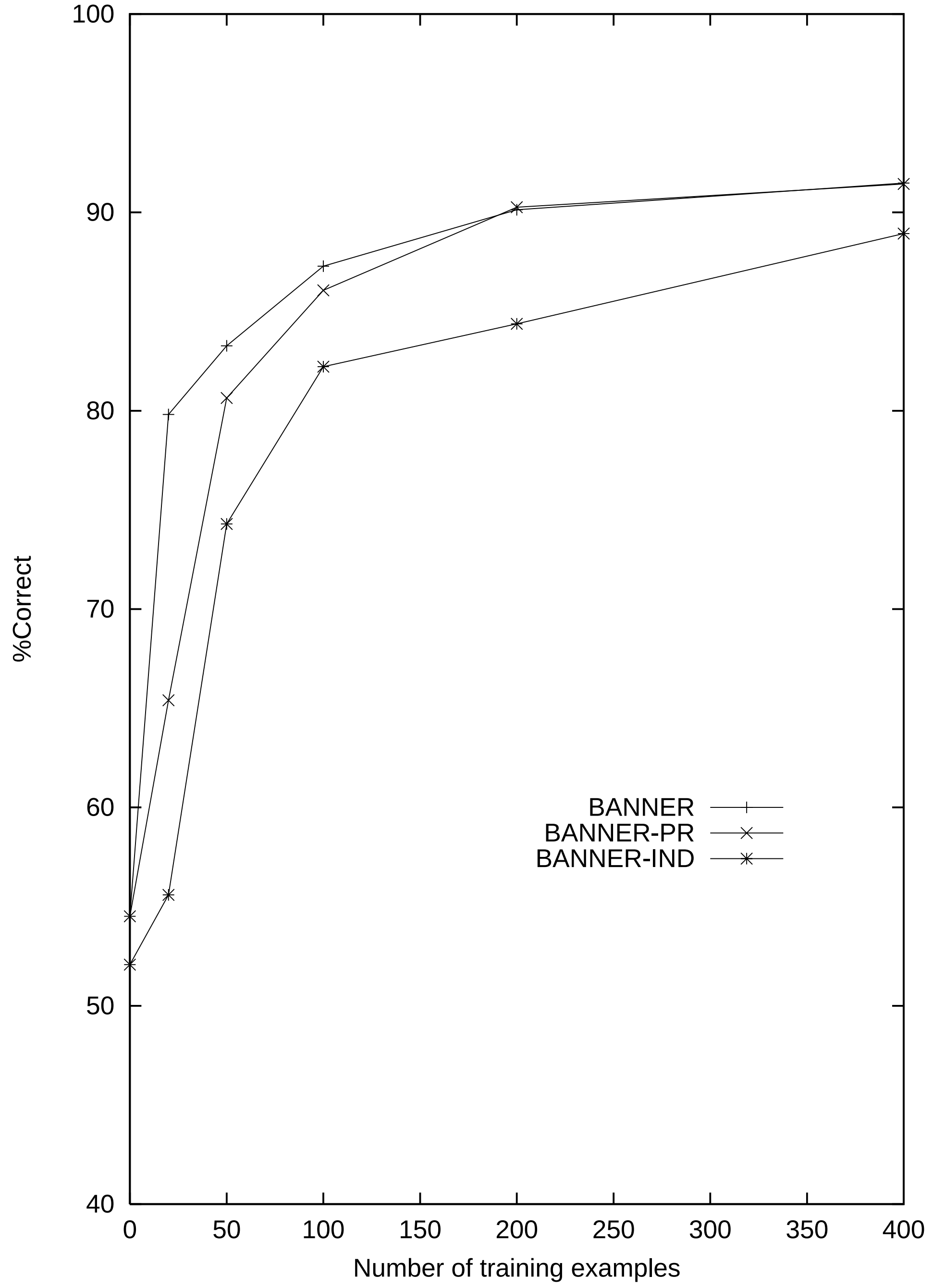


Figure 6.9: Splice-Junction: Banner Ablations

partially, and therefore their model of the domain must be close to the true model. Thus, it is expected that a theory refinement approach, starting with the true model of the domain, would lead to more accurate student models with fewer observations than learning these models from data alone.

Baffes (1994) originally evaluated this technique by performing controlled experiments on 100 students taking a C++ class. The students were divided into 4 groups of 25 students each. The students in each group were asked to take a pre-test and a post-test, each comprised of 10 questions. The groups differed in the technique used to generate feedback based on their responses to the pre-test questions. One group of students was given no feedback at all. The second group of students was given randomly generated explanations. The third and the fourth groups of students were given explanations using the student model learned by the tutor from their responses to the pre-test questions. These studies showed that the improvement in performance on the post-test was significantly higher for the students who were given feedback based on a student model. Thus, the ability to learn an accurate model of the students knowledge can have significant impact on the effectiveness of such a tutor. The tutor described in Baffes (1994) uses NEITHER (Baffes & Mooney, 1993), a logical theory refinement algorithm based on EITHER (Ourston & Mooney, 1994), to learn student models. Experiments were also performed to study the importance of the initial domain theory by comparing the student models learned by NEITHER with those learned by PFOIL, which is an inductive algorithm for learning propositional Horn-clause rules, based on FOIL (Quinlan, 1990).

The C++ tutor was intended to cover the following two concepts: the ambiguity involving statements with lazy operators, and the proper declaration and use of constants. The exercises presented to the students were multiple-choice questions, such that, given a short piece of C++ code, the students had to classify the code as one that would result in a *compile-error*, or one that is *ambiguous*, or one that is *correct*, or free of errors. Each pre-test and post-test consisted of 10 questions of which three were of type *compile-errors*, four were of type *ambiguous* and 3 were correct. The same pre-test and post-test were given to each student, although the order of the questions was randomized. Appendix B.3 shows the set of classification rules for this domain. The theory uses 14 input features that describe various aspects of C++ code. The value of some of these input features may be missing depending on the code. Figure 6.10 (Baffes, 1994) shows a short piece of correct C++ code, and its translation into a feature vector.

Baffes (1994) used data from the group of 25 students who received no feedback between pre-test and post-test to evaluate the effectiveness of NEITHER in learning accurate student models. In order to generate training and testing data splits that would be representative of the distribution of the classes in the pre-test and the post-test, 10 pairs of examples were generated by pairing up the examples in these tests that belonged to the same class. One example of each pair was randomly chosen to be included in the training set and the other was included in the test set. This gives rise to 2^{10} possible training splits for each student, of which 25 were randomly generated to be used in the evaluation. Thus, there were 25 training splits for each student, which resulted in 625 trials of the experiment. Each learning algorithm was run on each of the splits and the results were averaged over the 625 trials.

Our rationale for choosing this domain for evaluating BANNER is that, while the rules governing the correctness of C++ programs may be logical and deterministic, students may not have

```
CODE:
          void main()
          {
            const int j = 3, *h;
            int i,k;

            h = &j;
            cin>>k>>i;

            cout<<(k%j); cout<<(i%=j);
          }

FEATURE VECTOR:

  (pointer non-constant) (integer constant) (pointer-init false)
  (integer-init true) (integer-set no) (multiple-operands false)
  (position-a normal) (operator-a-lazy ?) (left-a-value ?)
  (on-operator-a-side right) (on-operator-b-side right)
  (operator-a-modify-assign) (operator-b mathematical)
```

Figure 6.10: Example C++ code and corresponding feature vector

such a clear cut understanding of the rules. Thus, the probabilistic aspects of Bayesian networks may help in modeling a student's confusion or lack of complete knowledge of the domain. We evaluated the performance of BANNER on this data set by running 625 trials, using the training/test splits used to evaluate NEITHER and PFOIL. Due to the presence of missing values, these experiments used C-APN for parameter revision. This was a viable option for this problem because the initial network was smaller, involving fewer variables, than the networks for the previous domains, and the number of training examples in each trial was very small. Tables 6.1 and 6.2 summarize the results. We did not generate learning curves for this domain because the size of the data per student is very small (20 examples).

System	**Average Accuracy**
NEITHER	62.0
BANNER	61.7
CORRECT THEORY	55.8
NAIVE BAYES	46.6
PFOIL	49.4

Table 6.1: C++ Student Modeling: Performance of Various Systems

The use of C-APN for parameter revision, and the large number of examples with missing data makes this experiment different from the previous experiments. The results demonstrate the feasibility of using C-APN with BANNER. They also show that BANNER is successful in revising a theory, as well as learning inductively, in the presence of missing data. It is interesting to note that its performance is very close to, but not better than that of NEITHER. Our conjecture is

System	Average Accuracy
BANNER	61.71
BANNER-PR	62.2
BANNER-IND	44.2

Table 6.2: C++ Student Modeling: BANNER Ablations

that the student models learned by BANNER would be more useful for purposes of remediation, since they provide a more quantitative model of the relative strengths and weakness in a student's understanding of the domain. Whether or not this conjecture is valid is an open question.

The difference in performance between BANNER and BANNER-PR is not statistically significant. Thus, BANNER-PR performs comparably with BANNER, which indicates that, for this data set, one can get as much mileage by adding a probabilistic component to the initial logical theory as by modifying it symbolically. However, combining structure revision with parameter revision does not seem to buy improved accuracy. One explanation for this is that, in many of the trials, revising the structure to fit the training data resulted in *overfitting*. Table 6.3 shows the accuracies of the three versions of BANNER on the training data. The differences in the training accuracies of these systems are all statistically significant. The results indicate that BANNER was able to learn the training data almost perfectly, but it did this at the cost of generalization. This is not very surprising in this domain, since failure to understand a concept can lead to inconsistent responses from students. The fact that this data set is impoverished in terms of the number of training examples available for each student model precluded the possibility of applying any of the techniques used to avoid overfitting, many of which require that some of the training data be held back.

System	Training Accuracy
BANNER	99.95
BANNER-PR	90.50
BANNER-IND	93.21

Table 6.3: C++ Student Modeling: Training Accuracies

Finally, it should be noted from the tables above that, because BANNER starts with an initial approximate theory, it is able to produce significantly more accurate models (significant at the 0.001 level) than NAIVE BAYES or PFOIL.

6.5 Chess End-games: Induction from Many Examples

An end-game is a situation in a game of chess where very few pieces remain on the board and only a short sequence of moves are needed to end the game in a win, a loss, or draw. Here, we are concerned particularly with King+Rook (White) versus King+Pawn on a7 (Black) end-games. The task is to learn to recognise whether a given board position can lead to a victory for White, assuming that it is White's turn to move. First used by (Shapiro, 1983, 1987), this data set consists of 3196 examples, where each example encodes various features of a board position, and is tagged with a label that indicates whether or not that position can lead to a *win* for White. Of these 3196 examples, 1669 (52%) are classified as *win*, and 1527 (48%) are classified as *no win*. Each example uses 36 discrete-valued features to describe board positions, of which 35 features have two values, and one has three. The data set is complete and has no missing values.

Although there are several problems that are suitable for studying the performance of Banner-Ind, we chose this problem for the following reason. Recent experimental analysis on various data sets (Kohavi et al., 1997), including the one under consideration, shows that the Naive Bayes algorithm performs significantly poorer than C4.5 (Quinlan, 1993) when the data consists of a large number of examples, and that this difference is particularly pronounced on the chess end-games data set. The purpose of this experiment is to study the behaviour of Banner in a situation where the Naive Bayes algorithm has been shown to perform poorly.

Since there is no initial theory associated with this domain, Banner has to be provided with a default initial theory, as discussed in section 5.6.1. Here, we present the results from two experiments: one where the initial network given to Banner was completely disconnected i.e. none of the nodes representing the input features and the target class are connected (Figure 6.11), and one where the initial network had all the features nodes connected to the target class node (Figure 6.12). We will refer to the former initial theory as the *disconnected* theory and the latter as the *connected* theory. Each of these experiments used a *3-fold cross-validation* methodology, where the data set was partitioned into three equal subsets with 1066 examples each, and the results were averaged over three trials, each using two of the above subsets of training and the remaining one for testing.

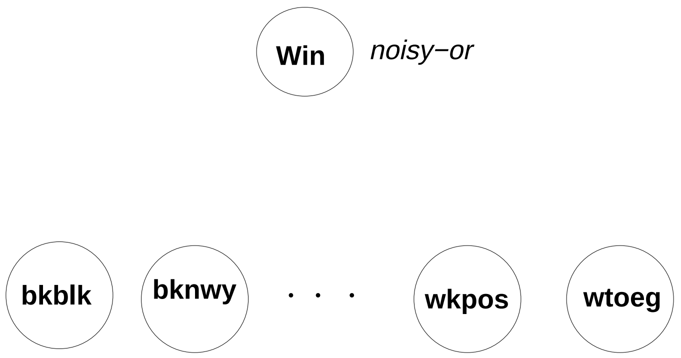


Figure 6.11: Disconnected initial network for the chess domain. Nodes *bkblk*, *bknwy*, *wkpos*, and *wtoeg* represent input features, and node *win* represents the class variable.

Table 6.4 compares the performances of Banner-Ind with that of Naive-Bayes on this data set. Kohavi et al. (1997) report that C4.5, when trained with 2130 examples, achieves an accuracy of 99.5%. The results show that Banner-Ind learns significantly more accurate classifiers for this domain than Naive-Bayes. The differences in accuracies, as shown in the table, are statistically significant at the 0.05 level or less.

Size is one of the main differences between this data set and the ones discussed earlier. Experiments on the DNA promoter recognition problem, DNA splice-junction recognition problem, and the student modeling problem involved less than 500 training examples, considerably less than

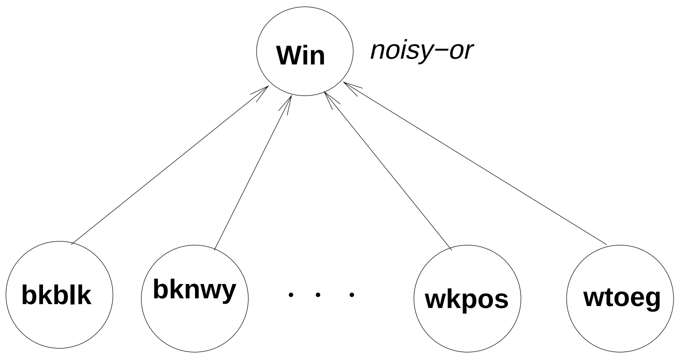


Figure 6.12: Connected initial network for the chess domain. Nodes *bkblk*, *bknwy*, *wkpos*, and *wtoeg* represent input features, and node *win* represents the class variable.

the number of training examples used here. Studies have shown that the Naive Bayes algorithm, with its high bias, and low variance, is more effective on smaller data sets with a few hundred examples, than on larger data sets with thousands of examples (Kohavi et al., 1997). BANNER-IND, on the other hand, explores a larger space of network structures than NAIVE BAYES, and therefore has a lower bias and higher variance. Based on the bias-variance difference between the two algorithms, it is expected that BANNER-IND will perform better than NAIVE BAYES on larger data sets. This is verified by our experiments, where NAIVE BAYES learned classifiers comparable to or better than those learned by BANNER-IND on problems with a few hundred training examples, such as the problems discussed previously, but was overtaken by BANNER-IND on this problem involving a substantially larger amount of training data.

It is interesting that using a fully connected network as the initial theory allows BANNER to learn networks that are more accurate than those learned using an entirely disconnected network. This indicates that BANNER is better at revising networks with extra links, than it is at revising networks with fewer links than necessary. In the former case, the parameter revision component is very effective in reducing the influence of the extra links to insignificance. However, the contribution of the structure revision component is still significant, since without it, BANNER-PR could only achieve an accuracy of 79%. The relatively poor performances of NAIVE BAYES, and BANNER-PR with the fully connected network, indicate that this problem requires a deeper structure than provided by these networks. In the experiments with the *disconnected* theory, BANNER seemed to get stuck in local minima from which it could not recover.

System	**Average Accuracy**
BANNER using the *connected* theory	95.31
BANNER-PR using the *connected* theory	79.02
BANNER using the *disconnected* theory	90.43
NAIVE BAYES	87.32

Table 6.4: Chess End-Game: Comparison of Various Systems

It is also interesting to note that the performance of our technique falls between that of

C4.5 and the Naive Bayes algorithm, with the C4.5 showing the best performance. Recall that C4.5 performed worse than Naive Bayes on the DNA promoter recognition and the DNA splice-junction recognition problems, while the performances of both Banner and Banner-Ind fell in between. In fact, Banner resulted in classifiers that were more accurate than those learned by the Naive Bayes algorithm most of the time. These observations indicate that Banner, by combining the ability to learn structured networks in a manner similar to logic-based learning algorithms, and the ability to learn a probabilistic representation, much like the Naive Bayes algorithm, forms a good compromise between the two approaches.

6.6 Brain Disorders: Revising an Abductive Theory

All the domains used for evaluating Banner so far have involved causal prediction tasks. However, Bayesian networks are used extensively in domains, such as medical diagnosis, that involve abductive or diagnostic predictions. For example, in the domain of medical diagnosis, Bayesian networks are used to represent causal models relating medical disorders to observable symptoms, and it is often necessary to predict disorders based on observations of symptoms, which involves abductive reasoning. Recall that Banner, when used with C-APN can be applied to such tasks. Here, we evaluate Banner on one such task of diagnosing brain damage due to a stroke (Tuhrim et al., 1991; Thompson, 1993; Thompson & Mooney, 1994). This task involves twenty-five different disorders, each representing an area of the brain areas that can be damaged (e.g. left frontal lobe, right temporal lobe), and thirty-seven observable symptoms (e.g. gait type), each with an average of four values. The data set consists of fifty examples, with an average of 8.56 symptoms and 1.96 disorders per example. This domain differs from the other domains examined so far in that the target classes are not mutually exclusive. Thus, it is possible for multiple disorders to be present simultaneously. This data set is accompanied by an abductive domain theory with 648 rules.

The given domain theory results in a very large Bayesian network where some nodes have as many as twenty parents. Recall that C-APN uses the conditional probability tables to revise the parameters of the networks. The sizes of the conditional probability tables ($\sim 2^{20}$) in such a large network makes it infeasible to use C-APN to revise this theory. Therefore, we have used a subset of this data set that includes only *four* of the twenty-five disorders, namely *left-frontal-lobe*, *left-temporal-lobe*, *left-parietal-lobe*, and *left-internal-capsule*. The reduced data set has an average of 1.04 disorders per example.

So far, we have only been examining domains where the target classes were mutually exclusive. For such domains, each example can belong to only one class, and classification accuracy is be measured by simply computing the percentage of examples that are correctly classified. When target classes are not mutually exclusive, i.e. when examples can belong to more than one class, the accuracy measure would have to be modified to account for partial correctness with respect to each example. Several accuracy measures have been proposed for such problems (Kulikowski & Weiss, 1991; Thompson, 1993; Thompson & Mooney, 1994), of which we will describe the following: *standard accuracy*, *intersection accuracy*, *specificity*, and *sensitivity*. Here, we will describe these measures in terms of the diagnosis domain, although they are equally applicable to any multi-class domain. Let C^+ be the number of disorders in the correct diagnosis of an example, C^- be the number of disorders not present in the correct diagnosis of the example, and S^+ be the number of disorders in the diagnosis generated by the classifier. Similarly, let T^+ (True Positives) be the number of disorders in the correct diagnosis of the example that are also in the diagnosis generated by the classifier, and let T^- (True Negatives) be the number of disorders that are excluded from both correct diagnosis and the diagnosis generated by the classifier. The *standard accuracy* for the example, defined as $\frac{(T^+ + T^-)}{(C^+ + C^-)}$, is the total set of disorders that are correctly predicted as present or absent. *Intersection accuracy* is defined as $\frac{(\frac{T^+}{C^+} + \frac{T^+}{S^+})}{2}$ and measures the percentage of disorders in the correct diagnosis that are correctly predicted, averaged with the percentage of the disorders generated by the classifier that are correct. *Sensitivity*, defined as $\frac{T^+}{C^+}$, measures accuracy at predicting the disorders actually present, while *specificity*, defined as $\frac{T^-}{C^-}$, measures the accuracy at excluding disorders not actually present. Of these, standard accuracy is the measure that is the closest to the accuracy measure used for domains with mutually exclusive classes. However, for problems where the number of possible disorders is much greater than the number of disorders actually present in an example, it is possible to get very high standard accuracy and specificity just by predicting that none of the examples have any disorders. Thus, these measures do not reflect the effectiveness of the classifier in identifying disorders. Similarly, it is possible to get perfect sensitivity by predicting that all the examples have all the disorders. Intersection accuracy avoids these extremes and is a good measure of the true effectiveness of the classifier in predicting disorders. This is the measure we use in our evaluation of various techniques on this domain.

The naive Bayes algorithm assumes that the target classes are mutually exclusive. In order to evaluate it on this domain, we constructed four independent naive Bayes classifiers, one for each disease, and each test example was classified by all the four classifiers to produce a diagnosis. Such a representation assumes that the probability of each symptom given a disorder is independent of its probability given any of the other disorders.

Reducing the number of disorders specified in the data without modifying the list of symptoms associated with the examples leads to a situation where several of the symptoms can no longer be explained by any of the disorders specified. This makes the data incomplete in the sense that causes for several of the input features are unknown. Note that this incompleteness is quite extreme since 21 of the 25 disorders have been removed from the data set. In order to model this incompleteness, BANNER, and all its ablated versions, were instructed to learn networks with leak nodes (as discussed in Section 5.6.2), thus allowing for the possibility of modeling unseen causes of symptoms.

All our experiments used the 10-fold cross-validation methodology described in Section 6.1. Figure 6.13 shows the performances of BANNER, BANNER-PR, BANNER-IND and NAIVE BAYES when trained with increasing numbers of examples.

The accuracy of the network derived from the initial theory, enhanced with leak nodes, is 74%. Without such enhancement, the network derived from the initial theory, since it is missing several important disorders, was found to be inconsistent with the data. The learning curves show that BANNER is successful in improving the accuracy of the initial theory by 8 percentage points. This experiment demonstrates that BANNER can be successfully applied to tasks involving diagnostic reasoning. In addition, it shows that BANNER can be used effectively even when the data is incomplete to a high degree.

Interestingly, the initial network used by BANNER, and the default initial theory used by BANNER-IND have identical accuracies. Both these theories predict that none of the examples

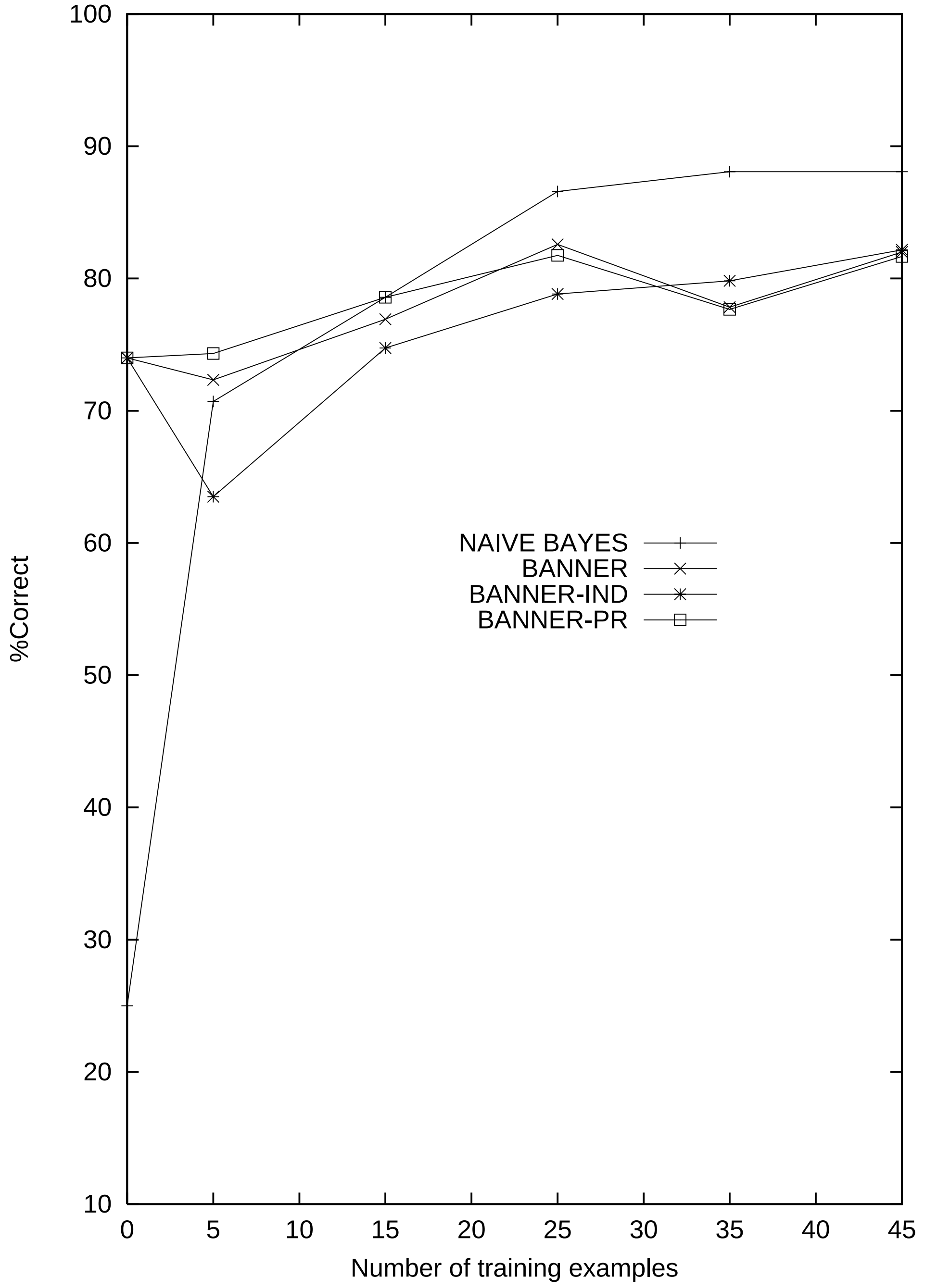


Figure 6.13: Diagnosing Brain Disorders: Performances of Various Systems

exhibits any of the four disorders. Training these networks with only 5 examples results in reduced accuracy, although this effect is not as pronounced for BANNER and BANNER-PR. An examination of the training/test splits showed that, for such small training sets, the distribution of the training examples is often very different from that of the test examples, so that an improved accuracy on the former results in reduced accuracy on the latter. However, the accuracy of the network starts to improve with increasing numbers of training examples (except at 35 examples). Note that, although BANNER-IND does worse than BANNER-PR and BANNER initially, it catches up with them when trained on as few as 35 examples. In addition, its learning curve is steadier than those of BANNER and BANNER-PR, and doesn't dip at 35 examples. None of the differences between BANNER, BANNER-PR, and BANNER-IND is statistically significant.

NAIVE BAYES outperforms BANNER on training sets with more than 5 examples, although the differences are statistically significant only at 35 examples (at the 0.02 level). A most likely explanation concerns the differences in the representations learned by these two techniques. NAIVE BAYES learns four independent classifiers, with the result that the diagnosis of the disorders are independent of each other. BANNER models the symptoms as Noisy-Or nodes, so that the influences of the disorders on the symptoms interact with each other, and hence their diagnoses are no longer independent. Such a representation is much richer because it models interactions, not only between each disorder and symptom, but also among the disorders and the symptoms themselves. It is expected that BANNER would need more examples than NAIVE BAYES since it learns a richer representation. The amount of data available for this problem is too limited to capture all these interactions accurately. Another consequence of this difference in the representations is that the high degree of incompleteness in the data, resulting from the removal of 21 of the 25 disorders specified in the original data, affects BANNER more than it affects NAIVE BAYES.

6.7 Evaluation of the Structure Revision Component

The experiments described in this sections were designed as controlled studies of the behaviour of the structure revision component. The idea is to corrupt known domain theories, and observe the ability of the structure revision algorithm to recover from these corruptions. We would expect that corrupting the domain theory would degrade the accuracy of the theory and that revising the structure of the theory would help recover from this degradation. Such experiments are commonly used to evaluate theory refinement algorithms (Pazzani & Brunk, 1993). We chose the *DNA promoter recognition* problem for these experiments because our earlier experiments indicated that the structure of the network, as specified in the initial domain theory, is more or less accurate. For each of these experiments, we corrupted the initial logical domain theory and used it to initialize BANNER NA BANNER-PR, which were then trained on the data set with 468 examples.

Although, it is common practice to corrupt theories randomly (Pazzani & Brunk, 1993), we found that the redundancy in the domain theory for DNA promoter recognition makes it very robust with respect to small corruptions. In order to study the behaviour of the structure revision component, we had to ensure that the corruptions to the theory degraded it to the extent that parameter revision alone would not be sufficient to recover from the damage. Therefore, we generated two corrupt theories by deleting portions of the original theory we knew to be critical. The two corruptions are progressively severe, and span the spectrum between the original theory and the empty theory (as used by BANNER-IND), and so allow us to study the performance of BANNER when given initial theories with increasingly inaccurate structure. The following subsections discuss these experiments further.

6.7.1 Deleting Intermediate Concept *Minus*_35

The first experiment studies the effect of deleting the intermediate concept *minus*_35 from the original theory for recognizing promoters, which results in the theory shown in Figure 6.14. Figure 6.15 shows the Bayesian network derived from this theory, henceforth called the *corrupt1* theory.

promoter	⟵	minus_10, conformation
minus_10	⟵	(p-14 t) (p-13 a) (p-12 t) (p-11 a) (p-10 a) (p-9 t)
minus_10	⟵	(p-13 t) (p-12 a) (p-10 a) (p-8 t)
minus_10	⟵	(p-13 t) (p-12 a) (p-11 t) (p-10 a) (p-9 a) (p-8 t)
minus_10	⟵	(p-12 t) (p-11 a) (p-7 t)
conformation	⟵	(p-47 c) (p-46 a) (p-45 a) (p-43 t) (p-42 t) (p-40 a) (p-39 c) (p-22 g) (p-18 t) (p-16 c) (p-8 g) (p-7 c) (p-6 g) (p-5 c) (p-4 c) (p-2 c) (p-1 c)
conformation	⟵	(p-45 a) (p-44 a) (p-41 a)
conformation	⟵	(p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t) (p-16 t) (p-15 g) (p-1 a)
conformation	⟵	(p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t) (p-21 a) (p-20 a) (p-17 t) (p-15 t) (p-4 t)

Figure 6.14: Domain theory corruption: The *corrupt1* theory

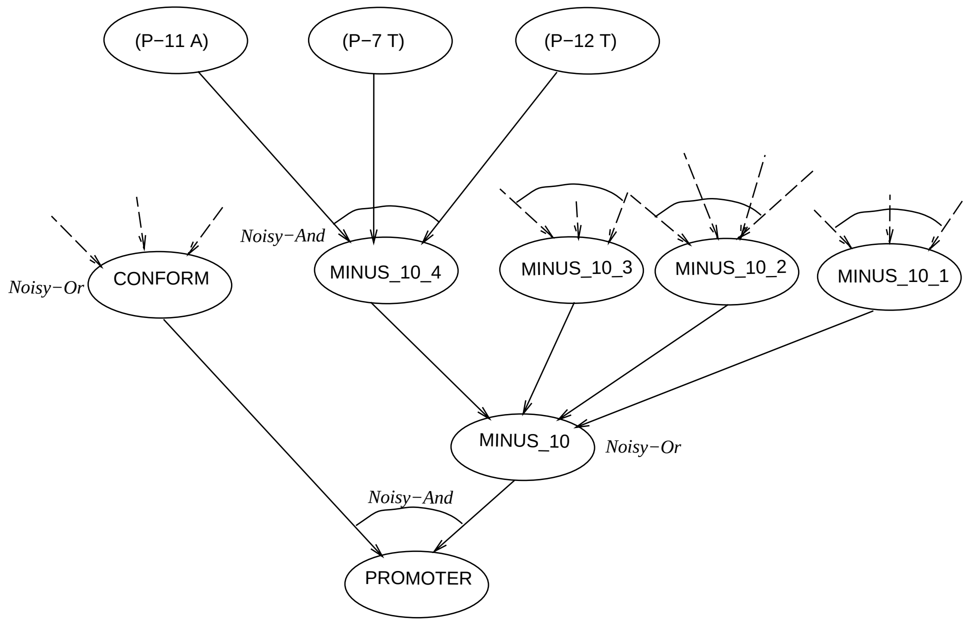


Figure 6.15: Bayesian network derived from *corrupt1*

Figure 6.16 shows the performances of BANNER-PR and BANNER with the *corrupt1* theory compared against the accuracy achieved by BANNER with the uncorrupted theory. The learning curves were generated using the *re-sampling* methodology with 20 trials, each with the same train/test splits used to evaluate BANNER on the original theory (Section 6.2.2). The graph shows that removing the intermediate concept *minus*_35 degrades the theory to the extent that, for most points in the learning curve, parameter revision alone cannot recover the accuracy attained by the original theory. Surprisingly, however, the corrupted theory results in better classifiers than the uncorrupted theory for very small training sets, indicating that the uncorrupted theory overfits these training sets. The learning curve for BANNER on the corrupted theory shows that, for larger training sets, the structure revision component is effective in recovering much of the accuracy lost due to the corruption, although this effect is statistically significant (at the 0.05 level) only for training sets with 400 examples.

The fact that BANNER and BANNER-PR result in comparable accuracies for smaller training sets can be partially explained by the fact that none of the trials with 10 and 20 training examples, and less than half the trials with 50 examples required structure revision. Notice that the corrupted theory results to better networks than the original, uncorrupted theory when trained on 10 examples. With 20 and 50 examples, the corrupted theory is still able to fit the training examples most of the time, without any need for structure revision, but results in poorer generalization. This leads to the hypothesis that, for smaller training sets, there are several theories that are as good as the original theory in fitting the training set, but are worse in terms of generalization, which would partially explain the observation that structure revision leads to improved training accuracies without any improvements in generalization, when trained on 50 and 100 examples.

Figure 6.17 shows an example of a revised network, with a view to highlight the modifications made by BANNER. The nodes and links added by BANNER are indicated as shaded ellipses, and thicker lines respectively. The numbers beside the new links indicate their weights. Note that, although some nodes have been replicated in the figure for clarity, there is no replication of nodes in the actual network. We see that BANNER has identified the following features to be added to the network: *P-35=T, P-36=T, P-34=G, P-33=A* and *P-3=A*. It has also added new links from the following features that are already a part of the network: *P-11=A*, and *P-10=A*. In addition, it has added three hidden variables, *I-1* through *I-3*. A comparison with the original theory indicates that, by adding features *P-35=A, P-36=T, P-33=A*, and *P-34=G* to the network, BANNER has successfully recovered some parts of the original theory that were left out. Furthermore, the logical relationship between these features has been preserved. However, in the original theory, these features combine conjunctively with the intermediate concept *minus*_10, whereas, here they combine disjunctively. That could explain why BANNER has added some of these features to the sub-network above *minus*_10_4. It has also identified some features not present in the original theory (*P-3=A*, *P-10=A*), although the links connecting these features to the network have low weights. Finally, note that the modifications to the network are spread out and not confined to any particular level. An examination of the revised networks from several trials revealed that the most common revisions are the addition of *P-35=T* and *P-36=T* to the network, and the modification of the links from *P-11=A* and *P-12=T* to *minus*_10_4.

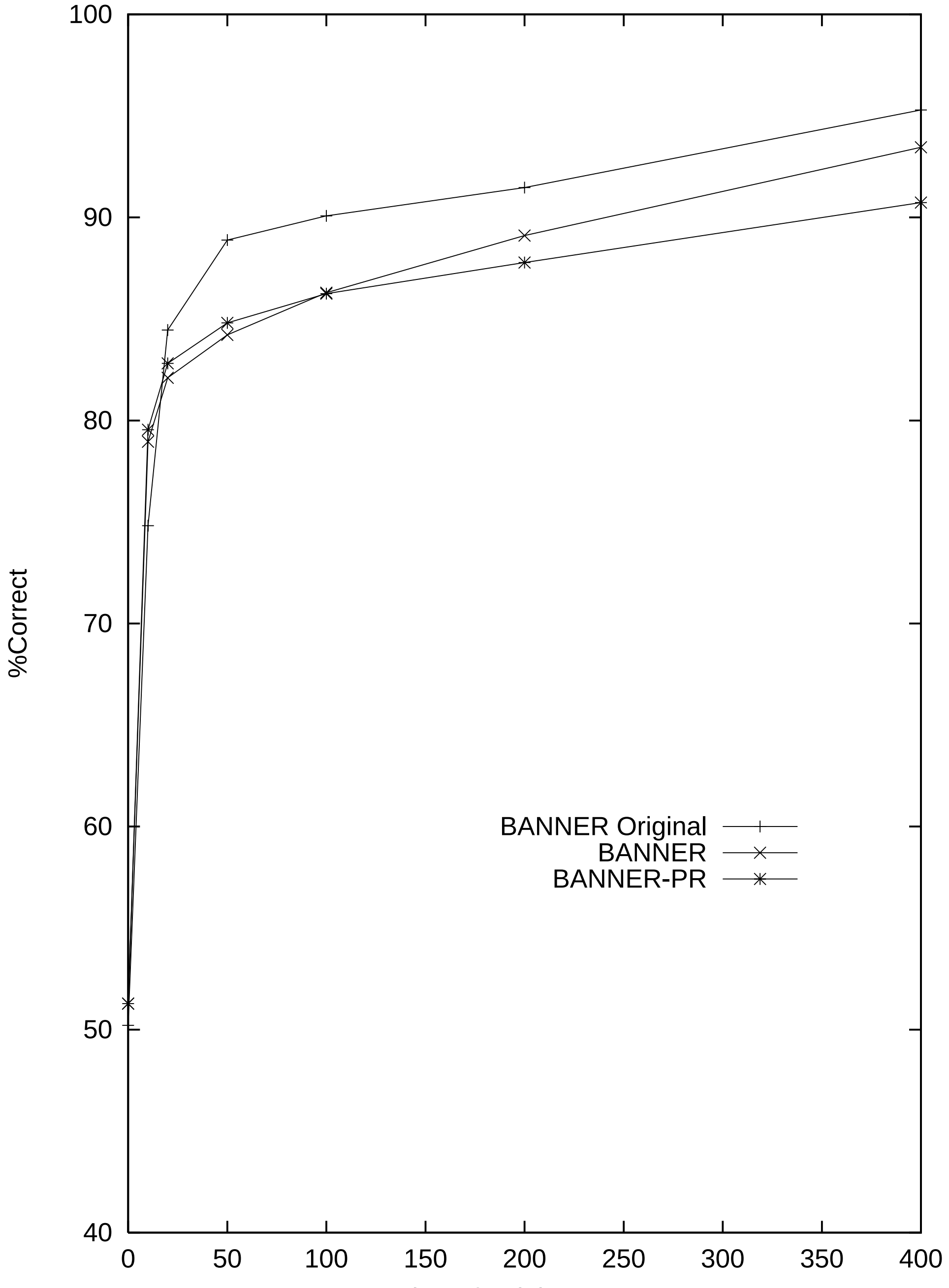


Figure 6.16: Effect of structure revision on *corrupt1*

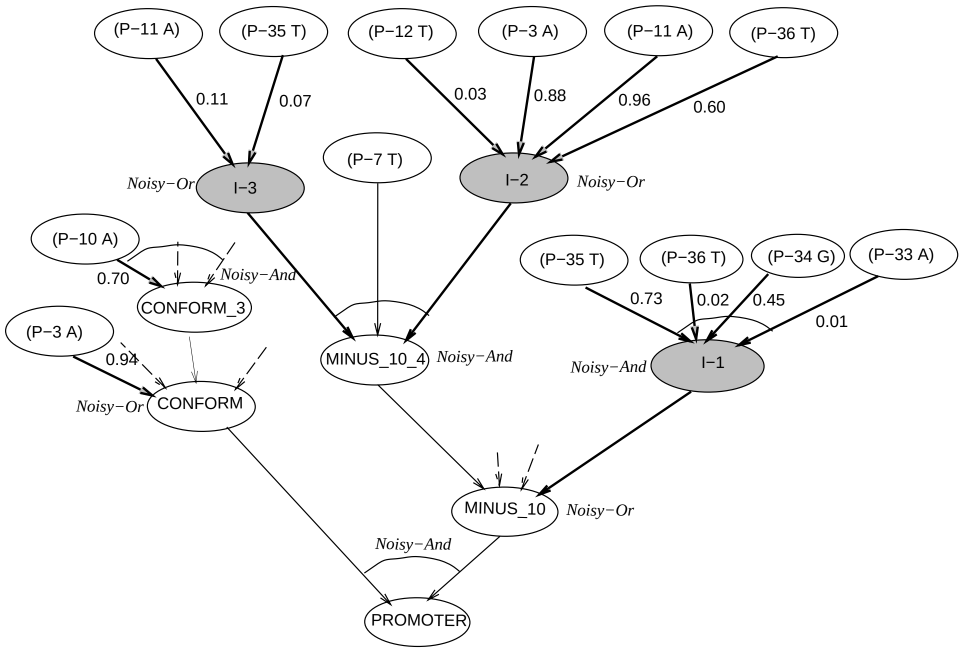


Figure 6.17: An example of a revised network derived from *corrupt1*

6.7.2 Deleting Intermediate Concept *Contact*

The theory shown in Figure 6.18, and henceforth referred to as the *corrupt2* theory, is derived by removing the intermediate concept *contact* from the original theory. This corruption is more severe than the one discussed in the previous subsection, and these two theories span the spectrum between that of not providing any initial theory, and that of providing the uncorrupted theory. Figure 6.19 shows the initial Bayesian network derived from this theory.

promoter	⟵	(p-47 c) (p-46 a) (p-45 a) (p-43 t) (p-42 t) (p-40 a) (p-39 c) (p-22 g) (p-18 t) (p-16 c) (p-8 g) (p-7 c) (p-6 g) (p-5 c) (p-4 c) (p-2 c) (p-1 c)
promoter	⟵	(p-45 a) (p-44 a) (p-41 a)
promoter	⟵	(p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t) (p-16 t) (p-15 g) (p-1 a)
promoter	⟵	(p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t) (p-21 a) (p-20 a) (p-17 t) (p-15 t) (p-4 t)

Figure 6.18: Domain theory corruption: The *corrupt2* theory

Figure 6.20 shows the accuracies of the final networks learned by BANNER-PR and BANNER using *corrupt2* as the initial theory compared with the accuracy attained by using the uncorrected theory as a starting point. That the deletion of the intermediate concept *contact* is a very severe corruption to the theory is brought out by the large gap between the accuracies of networks learned from the *corrupt2* theory and those learned from the uncorrupted theory (significant at the 0.001 level throughout). The graph shows that structure revision is effective in recovering more than 50% of the accuracy lost due to the corruption (significant at the 0.01 level throughout).

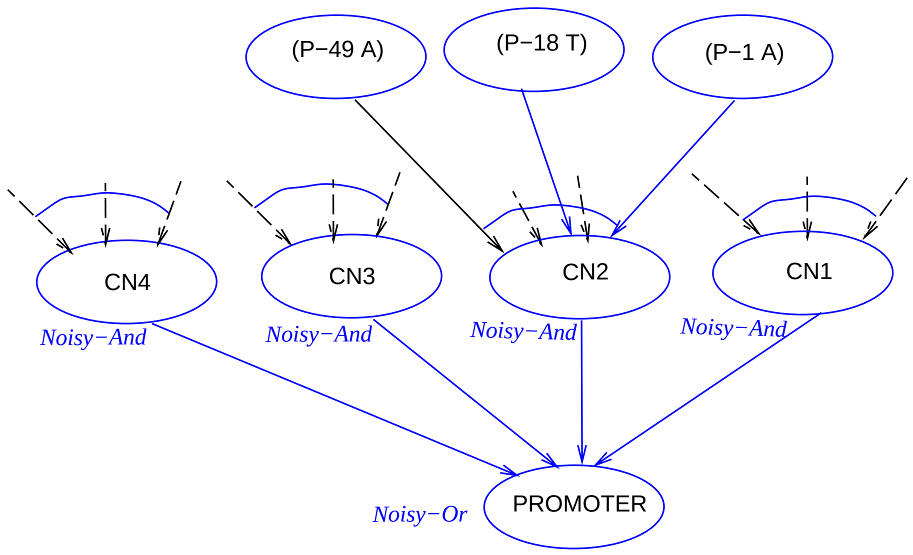


Figure 6.19: Bayesian network derived from *corrupt2*

Figure 6.21 shows an example of a revised network, where the nodes and links added by BANNER are indicated as shaded ellipses, and thicker lines respectively. The numbers beside the new links indicate their weights. Thus, we see that BANNER has identified that the input features *P-35=T, P-36=T, P-33=A, P-14=T, P-9=C* are necessary for classifying promoters. Referring to Figure 6.2, we see that all these features, except *P-9=C*, are indeed included in the original theory, and that several of the other deleted features have not been identified. BANNER has also added three hidden nodes, *I-1* through *I-3*. Considering the sub-network above node *I-1*, the logical relationships between the variables are similar to those defined in the original theory. Note that, while the modifications to the *corrupt*1 theory were spread out, the new nodes and links have been added as separate branch in this case. An examination of the revised networks generated by BANNER on all the trials revealed that it almost always added the features *P-35=T*, and *P-36=T* to the network. In fact, adding a link from *promoter* to *P-35=T* was, most often, the first revision to the initial network, resulting in a dramatic improvement in accuracy.

Finally, the graph in Figure 6.22 summarises the above results by comparing the performances of BANNER, given theories of varying degrees of corruption. At one end of the scale of corruption is the original uncorrupted theory, and at the other end is the empty theory, with the theories *corrupt1* and *corrupt2* falling in between. Note the revising *corrupt2* results in networks comparable in accuracy to learning from scratch. This indicates that *corrupt2* does not provide useful information about recognizing promoters, and in fact, many studies have noted that the intermediate concept, *conformation*, does not contribute significantly to this task (Ourston, 1991).

6.8 Summary of Results

In this chapter, we have presented the results of evaluating BANNER on the following real-world learning problems: recognizing DNA promoters (Noordewier et al., 1991), recognizing DNA splice-junctions (Noordewier et al., 1991), learning student models for a C++ tutor (Baffes, 1994),

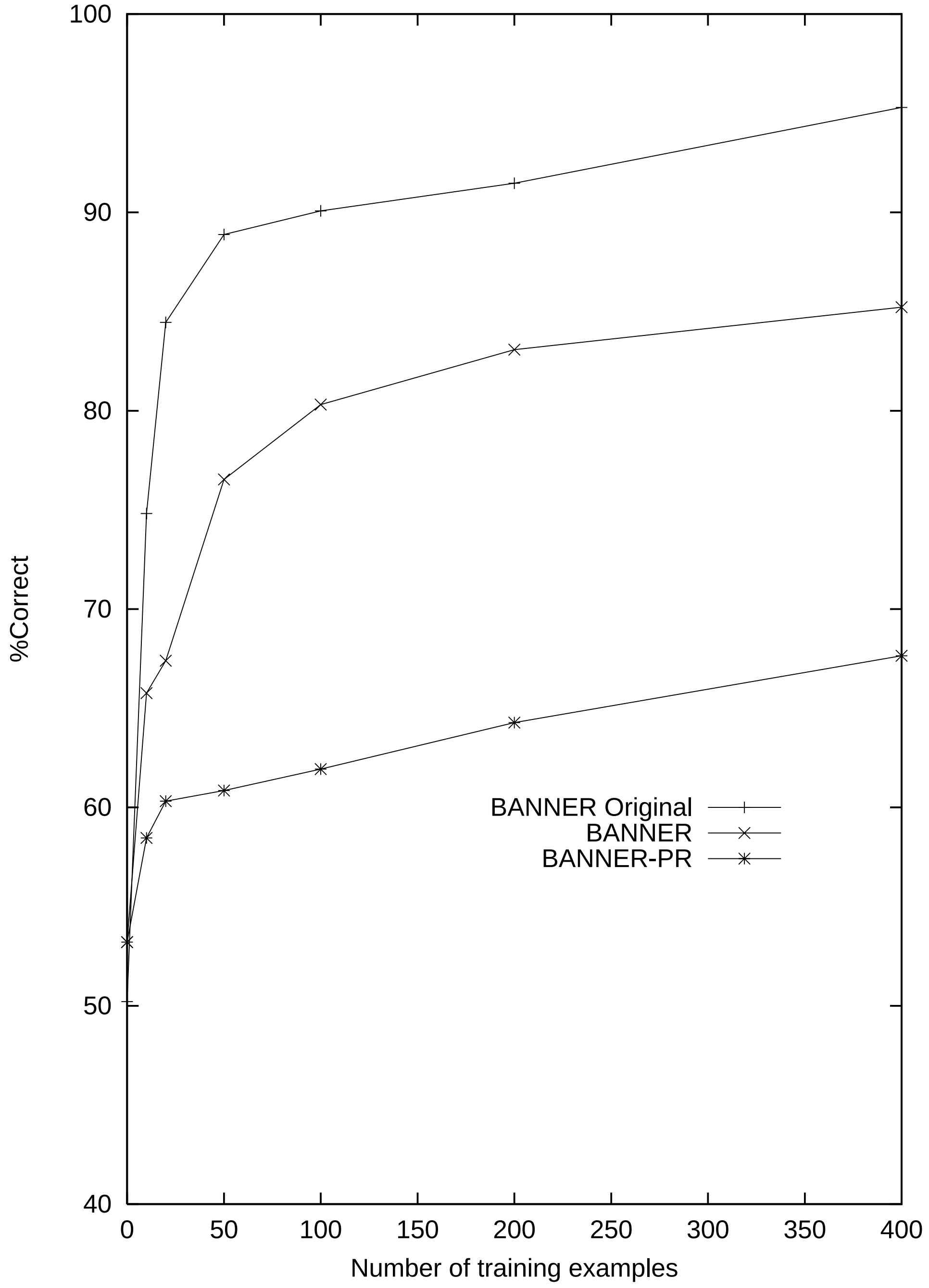


Figure 6.20: Effect of structure revision on *corrupt2*

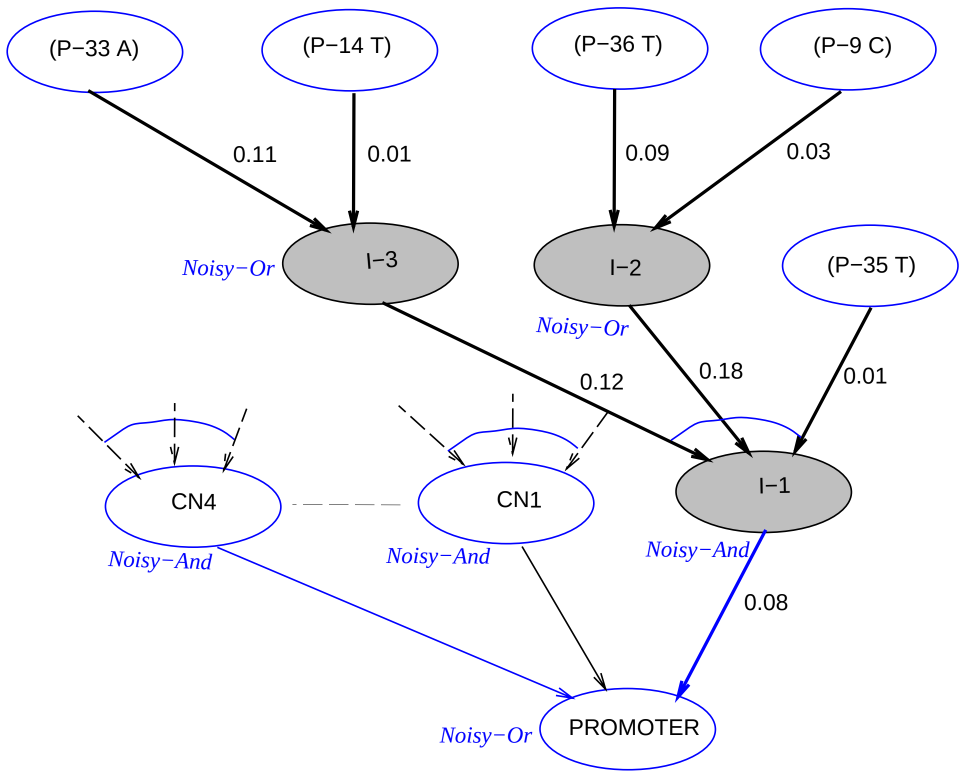


Figure 6.21: An example of a revised network derived from *corrupt2*

diagnosing brain disorders in human patients (Tuhrim et al., 1991), and classifying chess end-games (Shapiro, 1983, 1987). These problems present are diverse, and test different aspects of our technique. The first four of these problems, with their associated domain theories, test its ability to revise networks with hidden variables. The fifth problem evaluates how well it can learn in the absence of an initial domain theory.

The DNA promoter recognition problem, with the smaller data set, tests the effectiveness of our technique in using complete data to revise a given network intended for causal prediction. The larger data set of the same problem, and the *splice-junction* recognition problem present a slightly different learning scenario, where the data is incomplete. All of these problems, as well the chess end-games problem, provide a test-bed for evaluating BANNER-PR.

The problem of learning student models for a C++ tutor also tests the effectiveness of BANNER in revising networks for causal prediction, in the presence of hidden variables, and from incomplete data. However, since the data is incomplete in a way that violates the conditions of applicability of BANNER-PR, it acts as a test-bed for evaluating the effectiveness of using C-APN for parameter revision, and for demonstrating the generality of the structure revision algorithm. The domain of diagnosing brain disorders provides further opportunities to study the generality of BANNER by applying it to learn networks for non-causal (abductive) prediction.

These problems also provide variety in terms of the sizes of the data sets. The data sets associated with the chess end-games problem, and the C++ tutor problem are very small, with 50 and 20 examples respectively. The DNA promoter recognition problem, and the DNA splice-junction recognition problem form the middle-ground, with data sets of sizes ranging from 100

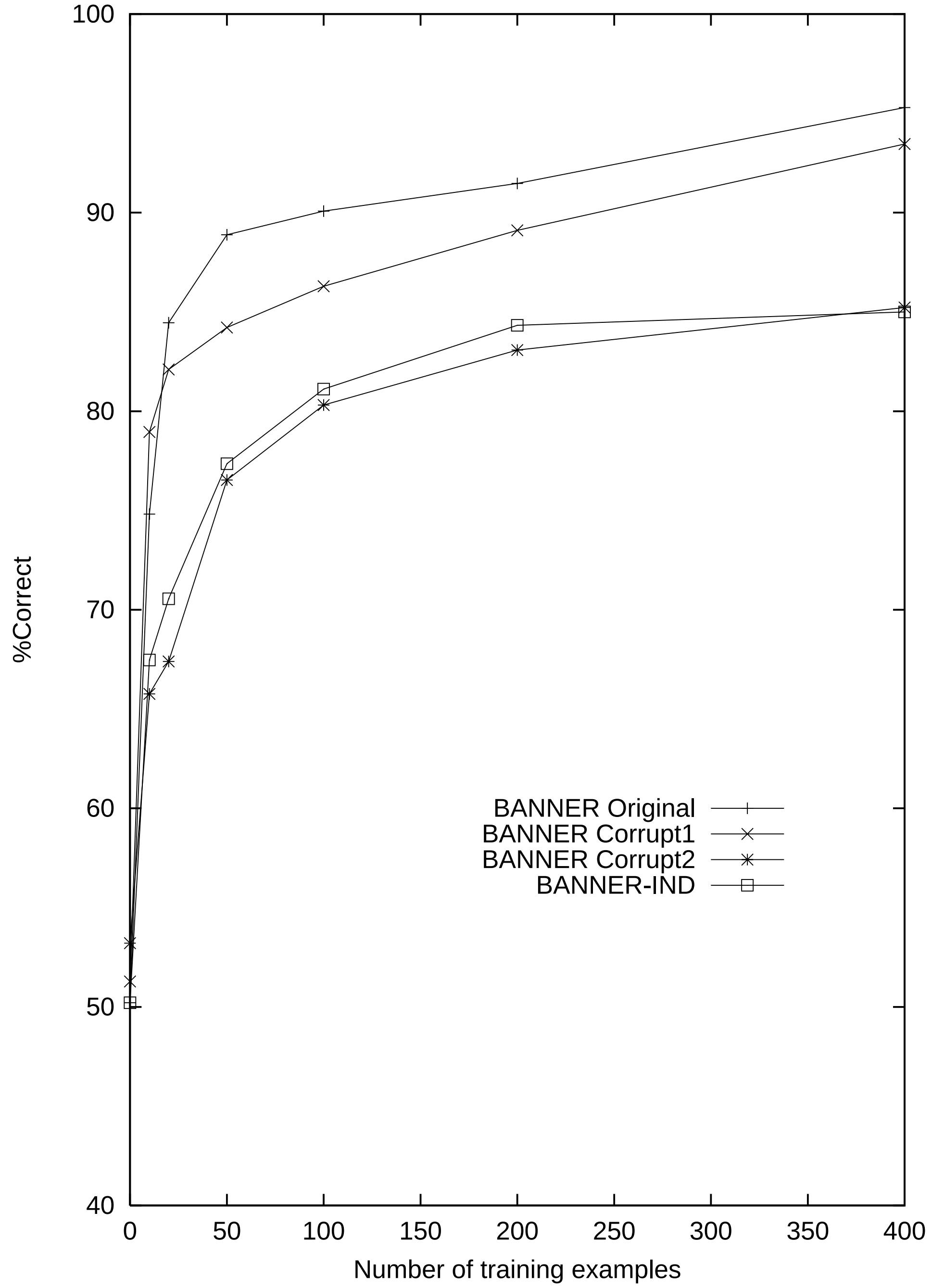


Figure 6.22: Effect of structure revision on theories of varying degrees of corruption

to 500 examples. Finally, the chess end-games problem provides a data set with more than 3000 examples.

The results of our experiments on all these domains demonstrate that BANNER is indeed effective in revising networks with hidden variables, resulting in improvements in accuracies ranging from 6% points on the C++ tutor domain, to about 40% on the DNA promoter recognition domain. They also show that BANNER can be usefully applied to a wide range of learning scenarios. Results on the problem of diagnosing brain disorders show that BANNER can, in principle, be used to effectively revise networks intended for diagnostic classification, although this issue needs to be explored further.

Many of the improvements in accuracies resulted from parameter revision, although the DNA splice-junction recognition problem clearly benefited substantially from structure revision for smaller training sets. Experiments on the C++ tutor domain indicate that overfitting is an issue in situations where only a limited amount data is available. Thus, on this domain, structure revision resulted in significant improvement in training accuracy as compared to parameter revision alone, without a corresponding improvement in generalization. That structure revision can contribute significantly towards improving the accuracies of network is demonstrated by the experiments on the DNA promoter recognition problem using corrupted versions of the initial domain theory. In addition, since the performance of BANNER-IND when given an initial empty theory is crucially dependent on the structure revision algorithm, the results obtained for the former on these problems, also reflect upon the latter. The fact that BANNER-IND can learn good classifiers for most problems, especially the chess end-games problem, where it learns highly accurate network starting from an empty theory, shows that the structure revision algorithm is effective. Finally, a comparison of the performances of BANNER and BANNER-IND shows that, for most domains, starting out with an initial, approximate network results in better classifiers than learning from scratch.

Comparing BANNER and the Naive Bayes algorithm on these problems, we see that starting with an initial, approximate theory gives BANNER a significant edge over Naive Bayes on smaller data sets. This effect is apparent on all problems, except on the problem of diagnosing brain disorders. Without an initial theory, BANNER-IND performs comparably with the Naive Bayes algorithm on the smaller data set of the DNA promoter recognition problem, but is worse on the rest of the problems, except on the chess end-games problem, where it performs substantially better.

Our evaluations show that BANNER performs comparably with the other hybrid theory refinement algorithms, RAPTURE and KBANN, on all problems. While it performs significantly better then the logical theory refinement algorithm EITHER on the DNA promoter recognition problem, it can only match the performance of NEITHER, a variant of EITHER, on the C++ tutor domain. Both BANNER and BANNER-IND significantly outperform logical inductive learning techniques on most problems, except the C++ tutor domain, where BANNER-IND results in poorer classifiers than PFOIL.

Finally, the experiments on the DNA promoter recognition problem, the DNA splice-junction problem, and the chess end-games problem demonstrate that, although restricted in its applicability, BANNER-PR is very fast, efficient and effective when applicable. None of these experiments would have been possible without it.

Chapter 7

Related Work

The research presented here is related to several recent advancements in the areas of theory refinement and learning Bayesian networks. In the following sections, we will place our research in the context of the existing body of research in these areas and highlight its contributions.

7.1 Learning Bayesian Networks

Research in the field of learning Bayesian networks has made rapid progress in recent years, and many difficult problems, such as learning from incomplete data, have been addressed. However, there have been few techniques, until now, for learning or revising Bayesian networks with hidden variables, and none of them have been shown to be efficient, effective, and scalable through evaluation on real problems. In this dissertation, we have presented one such technique for revising a class of Bayesian networks, namely those with Noisy-Or and Noisy-And nodes, which can revise networks with hidden variables as well as discover hidden variables when necessary. Rather than search through the space of all possible revisions, BANNER uses information in the data to select specific nodes and links in the network to be revised, which makes it efficient. The initial theory given to BANNER may be in the form of a Bayesian network with Noisy-Or/And nodes, or in the form of propositional Horn-clause rules. Thus, our technique also provides a direct mechanism for incorporating knowledge expressed as propositional Horn-clause rules into a Bayesian network. Although designed for theory refinement, BANNER can also be used as a system for learning Bayesian networks with hidden variables inductively from data. We have evaluated our system on several real-world problems that demonstrate its effectiveness in revising fairly large networks. We have also presented BANNER-PR, which is an algorithm for revising the parameters of a restricted class of Noisy-Or/And networks, in the presence of hidden variables, and have shown it to be computationally efficient, and effective in producing accurate classifiers. In the following subsections, we first compare BANNER-PR with some of the existing techniques for learning the parameters of a Bayesian network, followed by a discussion of how BANNER relates to other techniques for learning the structure of a Bayesian network.

7.1.1 Parameter Learning

Learning the parameters is fairly straightforward when all the variables of the network are represented in the data. A common approach is to use the maximum likelihood estimates for the parameters, which in the case of no hidden variables, reduces to a function of the relative frequencies of occurrences of the values of the variable.

Approximation methods like *Gibbs Sampling* (Geman & Geman, 1984) and EM (Dempster et al., 1977; Lauritzen, 1995) have been proposed to learn the parameters of a network from incomplete data. Both these methods require some initialization of the parameters. Using the current network, the algorithms then estimate the values missing in the data. Complete data thus obtained is sampled to compute new values for the parameters. These steps are repeated until some convergence criteria are met. The goal of these methods is to optimize the likelihood of the data given the network. APEM (Thiesson, 1995) is a technique that combines traditional EM with gradient descent methods for faster convergence. Preliminary experiments using APEM[1] on the DNA promoter recognition problem did not yield encouraging results (Ramachandran & Mooney, 1996a). Although APEM could improve the accuracy of the network, it could only achieve about 65% accuracy on the test sets. However, since APEM does not currently handle Noisy-Or or Noisy-And nodes, these nodes were modeled by general nodes with appropriate initial parameters. This results in the system being less constrained and having to learn significantly more parameters.

Díez (1993) proposes a technique where the parameters are represented as distributions, which are then refined incrementally based on the training examples. This technique is also shown to be applicable to Noisy-Or/And nodes. Like BANNER, it is also based directly on the computations used to propagate beliefs through a network, and hence suffers the same limitations in terms of the network architectures to which it can be applied. Moreover, this algorithm has not yet been implemented (Diéz, 1997), and since it makes certain assumptions about the distribution of the parameters, it is not possible to evaluate it without empirical studies. Musick (1994) also presents an approach where each parameter is represented as a distribution rather than as a point value. The focus of this research is less on induction of networks and more on inference techniques for Bayesian networks with parameters specified as distributions.

APN (Russell et al., 1995) is a technique that uses gradient descent to learn the parameters of a network given incomplete data. It can be used to learn a wide range of distributions, including Noisy-Or/And models. It is more general than BANNER-PR and can be applied to networks with loops. However, it does not exploit the linearity of computation offered by the Noisy-Or/And models. Our experiments, comparing the two techniques, show that APN, when applied to Noisy-Or/And networks converges much more slowly than BANNER-PR (see Section 4.3).

One of the early connectionist approaches to learning the parameters of a Bayesian networks is that reported in Neal (1992). It also uses the Noisy-Or approximation of a Bayesian node. However, since it uses stochastic networks similar to the Boltzmann machine, simulation of the network involves allowing the network to settle down to an equilibrium for each pattern observed. This is expensive and slows down learning dramatically. Our technique for parameter learning, BANNER-PR, uses a forward propagation algorithm which results in significantly faster training.

Schwalb (1993) addresses the problem of learning the parameters of a given Bayesian

[1]We used the implementation of APEM provided by Bo Thiesson.

network by mapping it onto a neural network with SIGMA-PI nodes and learning the conditional probabilities associated with the network (represented by link weights in the corresponding neural network) using standard backpropagation techniques (McClelland & Rumelhart, 1988). This has the advantage that it is able to learn the conditional probabilities even in the presence of hidden variables. However, the size of the neural network is combinatorial in the fan-in of the nodes in the corresponding Bayesian network, making the technique infeasible for even modestly large networks such as that for the DNA promoter recognition problem.

Kwoh and Gillies (1996) have recently proposed a technique similar to BANNER-PR for learning the parameters of a network with general, discrete-valued, probabilistic nodes. Like BANNER-PR, it performs gradient descent to minimize mean squared-error with respect to some output variables. Since, like BANNER they use the belief propagation computations for formulating the gradients, their technique faces the same limitations in terms on the kinds of network structures it can handle. As such, their computations are only valid for tree-structured networks and do not handle Noisy-Or/And nodes.

7.1.2 Structure Learning

Cooper and Herskovits (1992) have proposed a Bayesian approach to learning both the structure and the parameters of a Bayesian network. They view the problem as one of maximizing the probability of the network given the data, and propose a scoring metric that is used as a basis for hill-climbing incrementally through the space of networks to find one that is highly probable given the data. Others (Buntine, 1991; Heckerman et al., 1994) have followed up on this paradigm, introducing variations to improve the performance of the algorithm. Provan and Singh (1994) and Singh and Provan (1996) enhance this approach with techniques for restricting the number of attributes considered by the learning algorithm by selecting a small subset of relevant attributes, from the given set of attributes. All of these approaches assume that the data is complete. Recently, Ramoni and Sebastiani (1997) have extended this technique to learn from data with missing values. All of these approaches construct a Bayesian network incrementally. However, they use an undirected search through the space of possible incremental additions to the network, and spend considerable computational effort exploring unsuccessful paths. In addition to not being able to discover hidden variables, these algorithms are also not designed to learn models optimized for some particular task. Thus, the models they learn may be unnecessarily complex for their intended task.

MS-EM (Friedman, 1997) is a technique that extends the EM algorithm to learn the structure as well as the parameters of a network from incomplete data. Given a network and some data, it first iteratively modifies the parameters using the EM algorithm. When further modification of the parameters does not improve the network, it modifies the structure of the network using a technique similar to EM. These steps are repeated until convergence. This approach been shown to be effective in learning small networks from simulated data, but has not been evaluated on larger real-world data sets. Moreover, this technique has no mechanism for selecting a candidate set of nodes in the network that most likely need to be revised and, instead, relies on blind search through the space of all possible revisions. In addition, although this works even when the initial theory has hidden variables, it cannot discover any new hidden variables.

There are few practical, scalable techniques to add hidden variables to a Bayesian network. Connolly (1993) has proposed using a technique similar to COBWEB (Fisher, 1987) to build a

Bayesian network by clustering observed variables hierarchically into a tree. Observed variables form the leaves of the tree, and the interior nodes are hidden variables. However, this technique can only learn tree structured networks. Kwoh and Gillies (1996) discuss a procedure for adding hidden variables by first learning a Bayesian network from data using any inductive learning algorithm, and then using statistical analysis to find correlations between variables with the same cause and clustering such variables with a new hidden node. This technique uses the *generate then test* strategy, requiring the examination of the entire space of all possible clusterings of variables with common causes, and is computationally expensive. In addition, it has only been evaluated on very small networks with less than ten variables.

MS-EM is one of the few techniques that can be used for theory refinement. However, it has not been designed for theory refinement and lacks mechanisms to focus on portions of the network that are the most likely candidates for revision. Buntine (1991) has proposed a technique for revising a Bayesian network efficiently, using scoring metrics similar to that proposed by Cooper and Herskovits (1992). However, he does not specify any method for recognizing *when* the network needs to be revised. Nor does he discuss ways of focusing on the portions of the network that should be modified. As such, his approach involves extensive search and is therefore inefficient, especially in the presence of hidden variables. Lam and Bacchus (1994b) have a technique for incrementally refining a Bayesian network using the Minimum Description Length (Rissanen, 1978) principle. Their approach, however, can only modify those portions of the network whose variables are observable. Thus, it cannot modify networks with hidden variables, nor can it add hidden variables to the network.

In addition, many of the approaches discussed above provide a way of specifying prior knowledge about the structure of the network, in the form of prior probabilities of network structures. However, there has been no effort to study the effectiveness of using the above techniques for theory refinement. Most of the research in this field is focused on learning from scratch, and barring some exceptions (Buntine, 1991; Lam & Bacchus, 1994b), the issue of theory refinement has largely been ignored.

Apart from Schwalb (1993), none of the techniques mentioned above are concerned with learning networks that are optimized for their intended task. Singh and Provan (1996) have proposed a selective induction technique that learns Bayesian network classifiers by using information theoretic metrics to select a subset of attributes that are most relevant to the classification task, and using a variant of the K2 algorithm to learn a network with these attributes. Here, the focus is more on techniques for limiting the number of attributes to be considered by the learning algorithm to those that best predict the classification of the examples in the data, than on developing an induction algorithm that is geared towards optimizing classification accuracy. This technique for selecting a subset of relevant attributes may be used in conjunction with other learning algorithms. Friedman and Goldszmidt (1996) have proposed an algorithm for learning a restricted class of Bayesian networks, called *tree-augmented naive Bayes* (TAN), that are optimized for classification. TANs are extensions of the Naive Bayes classifier, such that each attribute has links from the class variable and at most one other attribute. This technique has been shown to be better than the Naive Bayes algorithm on several real-world data sets. However, it is not designed for theory refinement. Greiner et al. (1997) analyse the advantages of learning Bayesian networks that are optimal for the distribution of queries found in the data. They also propose a hill-climbing algorithm that uses this criteria to learn the parameters of a network. However, they do not present any experimental evaluation of their technique.

While there are several algorithms for learning Bayesian networks, experimental evaluation of techniques on real data sets are scarce. With the exception of a few (Provan & Singh, 1994; Singh & Provan, 1996; Friedman & Goldszmidt, 1996; Kwoh & Gillies, 1996), most techniques have been evaluated on artificial data generated from artificial networks, or from networks built to model certain tasks. Furthermore, there have been no evaluation for revising Bayesian networks on real problems, using real data. On the other hand, we have demonstrated the effectiveness of BANNER through extensive evaluation on real-world learning problems involving fairly large networks.

7.2 Theory Refinement

Early research in theory refinement focused on purely logical representations. Such representations are very comprehensible, but were found to be inadequate for many real-world problems. Classifiers produced via logical theory refinement were found to perform poorly on such problems. This led to the idea of integrating logical and numerical representations, and the development of algorithms to revise rule bases by mapping them to representations that employs some form of uncertain reasoning or numerical summing of evidence. Such techniques have been shown to be very effective in many domains, often resulting in dramatically improved accuracies. However, what these approaches gained in performance, they lost in comprehensibility. Such hybrid representations lacked well-defined semantics and could not be easily understood. BANNER, by using the Bayesian network representation, bridges the gap between performance and comprehensibility. Experiments show that BANNER performs as well as any of the other existing hybrid learning systems. At the same time, the networks produced by BANNER are easily understood within the framework of Bayesian networks. Another advantage of BANNER is that it can be used to learn networks for causal and well as evidential reasoning. Most existing techniques for theory refinement are geared towards classification tasks involving causal inference. Thus, they cannot be used in cases when the classification task involves evidential reasoning. Since BANNER is built upon Bayesian inference mechanisms, which can be used for deductive as well as abductive inference, it does not suffer from this limitation. In the following subsections, we first present a overview of the related research within logical theory refinement, and then discuss some of the hybrid learning algorithms.

7.2.1 Logic-Based Approaches

Mechanisms for picking appropriate revision points in a theory is one of the key aspects of logical theory refinement algorithms. While techniques like EITHER (Ourston & Mooney, 1994) and NEITHER (Baffes & Mooney, 1993) have mechanisms for tracing blame for misclassifications through the theory, these mechanisms do not provide them with any direct way of determining the level at which to revise the theory. Viewing the theory as an AND-OR graph, both EITHER and NEITHER first select the best revision at the lowest level of the theory. They then hill-climb up the path from the selected revision point to the root of the theory, evaluating appropriate revisions to each node in the path, and selecting the one that is the simplest. This adds to the computational cost of these techniques.

Probabilistic theory revision (PTR) (Feldman, Segre, & Koppel, 1991; Koppel et al., 1994) takes a different approach to revising propositional Horn-clause theories. This technique uses probabilities to express confidence in the correctness of different portions of the theory, which are then used to locate faults. PTR converts the logical theory into an AND-OR graph. Each edge in the graph is assigned a probability that reflects the prior confidence that the edge is a part of the target theory. PTR iterates over the examples to compute the posterior probabilities associated with the edges given the data. The network is revised whenever the posterior probability associated with an edge in the network falls below a certain threshold. It should be emphasized that, while PTR uses probabilities to locate faults in the theory, the theory itself always remains logical. Experiments on the DNA promoter recognition problem (Noordewier et al., 1991) show that the performance of PTR is only slightly better than that of EITHER.

7.2.2 Hybrid Approaches

There are several techniques that combine logical and numeric representations to learn accurate classifiers. One such technique is KBANN (Towell & Shavlik, 1994) which converts an initial logical domain theory into a multi-layered feedforward neural network and uses gradient descent back-propagation to improve the accuracy of this network on the data. The final network may itself be used as a classifier. Alternately, there are techniques for extracting logical rules out of the final network (Towell & Shavlik, 1993; Craven & Shavlik, 1996; Craven, 1996). While KBANN only refines the weights of the network, TOPGEN (Opitz & Shavlik, 1993; Opitz, 1995b) is a technique for modifying the structure of the network by adding new nodes and links. Thus, KBANN and TOPGEN are together comparable to BANNER. Our experiments show that the performance of BANNER is comparable to that of KBANN in terms of classification accuracy. However, a clear advantage of BANNER is that it learns Bayesian networks that have well-defined semantics. Thus, the final networks produced by BANNER are easily understood in terms of the semantics of Bayesian networks, and, unlike the networks produced by KBANN, do not need to be post-processed into some comprehensible form.

Our research was inspired by RAPTURE (Mahoney & Mooney, 1993, 1994), which converts an initial certainty factor rule-base into a multi-layered feedforward neural network and uses a gradient descent backpropagation algorithm to revise the certainty factors. It can also modify the structure of the network by adding new links, and by adding new hidden nodes using the UPSTART algorithm (Frean, 1990). However, RAPTURE and BANNER take very different approaches to the problem of structure revision. While BANNER uses the information in the current network to pick revision points, RAPTURE limits its revisions to the output nodes. Thus, RAPTURE always attaches new nodes and links to output nodes, and cannot make any modifications to deeper layers of the structure. Sometimes, lower level rules concepts participate in several rules defining higher level concepts, and modifying the rules associated with such lower level concepts would be more efficient and simpler than modifying all the higher level rules in which they participate. In terms of performance on real data-sets, RAPTURE performs as well as, and sometimes better than BANNER by small margins. However, the networks produced by BANNER have more clearly defined semantics because Bayesian networks are theoretically more well-grounded than certainty factors.

Chapter 8

Future Work

This research presents many avenues for further exploration with respect to both parameter revision and structure revision of Bayesian networks. Interesting directions for future research include exploring ways to extend BANNER-PR so as to make it more general in its applicability, and extending the structure revision algorithm to handle general nodes and multi-valued nodes. We would also like study the effects of incorporating, into BANNER, some of the other existing techniques for adding links and hidden variables to a Bayesian networks. In addition, we would like to perform more experiments to study various aspects of these techniques further. We elaborate on these ideas in the following sections.

8.1 Experimental Study

We have evaluated BANNER on several real-world learning problems. The initial theories for all these problems were expressed in the form of propositional Horn-clause rules. We could not obtain any real data sets where the initial theory is expressed in the form of a Bayesian network with Noisy-Or/And nodes. One important direction for future research is to evaluate BANNER on revising such theories. We are also interested in applying BANNER to real-world applications where a Bayesian network representation is desired. Such an evaluation would shed more light on the usefulness of BANNER as a tool for learning Bayesian networks.

Most of the problems used to evaluate BANNER, with the exception of the problem of diagnosing brain disorders, have involved causal prediction (Chapter 6). The purpose of evaluating BANNER on the brain disorder diagnosis domain was to show that, in principle, BANNER can be used to revise networks intended for non-causal (abductive) classification. The results show that BANNER can effectively revise such networks, although it does not perform as well as the Naive Bayes algorithm, and is limited by combination of a high degree of incompleteness in the data due to the removal of information about a large number of disorders, and a limited number of training examples. Since Bayesian networks are used for many tasks involving diagnostic prediction, such as medical diagnosis, we would like to evaluate the performance of BANNER further other real-world problems involving such non-causal prediction, especially problems with a large number of training examples.

An interesting issue that we have touched upon in this dissertation concerns the advantage of training a network to be optimized for the specific task for which it is intended, compared to training it to model the data without regard to the intended task. We have adapted an existing parameter learning algorithm, APN (Russell et al., 1995), that uses the latter approach, to create an algorithm, C-APN, that learns classifiers by modeling the distribution of a specific set of class variables conditioned on a specific set of evidence variables. While Friedman and Goldszmidt (1996) have experimentally demonstrated the advantages of a system targeted specifically at learning classifiers, our experiments with APN and C-APN, however, have not provided any conclusive evidence of the advantage of one approach over the other. We would like to study this issue further by evaluating these techniques on more data sets, and on problems involving non-causal predictions.

Finally, we would also like to further study the usefulness of BANNER as a hybrid theory refinement system. Our experiments have demonstrated that the performance of BANNER is comparable to the performances of other hybrid learning algorithms such as KBANN (Towell & Shavlik, 1994) and RAPTURE (Mahoney & Mooney, 1993). In the future, we would like to test the hypothesis that the networks generated by BANNER are more comprehensible than the representations learned by the other techniques. This can be done through studies where humans rate the representations generated by these techniques according to how easy they are to understand.

8.2 Extensions to BANNER-PR

As mentioned earlier, BANNER-PR can only be used to learn a certain class of networks, namely those composed of Noisy-Or/And nodes. BANNER-PR also requires that the network should be a *polytree* or a *virtual polytree*. Finally, it can only be used to learn networks intended for causal prediction tasks. BANNER-PR can be extended quite easily to the case of general nodes. The techniques proposed by Schwalb (1993) and, more recently, by Kwoh and Gillies (1996) are very similar to BANNER-PR in that they use a gradient descent, backpropagation approach to learning the parameters of a network with general nodes and can be easily incorporated into our technique.

Extending BANNER-PR to learn networks for non-causal classification tasks is also fairly straightforward. Gradient descent backpropagation involves two phases: a first phase involving the propagation of the effects of the evidence to the class variables, and the second pass involving the propagation of errors from the class variables to the rest of the network. Assuming causal reasoning, a single pass through the network is sufficient to propagate the influence due to the evidence. This means that a single backpropagation pass is sufficient to propagate the errors. For arbitrary inference tasks, the influence propagation phase would require multiple forward and backward passes through the network. Therefore, the error propagation phase would require multiple passes through the network as well. The computations of the gradients would be more complex. We have derived these computations, but have not implemented an abductive version of BANNER-PR because we do not have interesting data sets with which to evaluate it.

Extending BANNER-PR to handle networks with loops, however, presents a challenge. The errors computed during the backpropagation phase are based on the computations involved in belief propagation through the network. Most of the inference techniques applicable to networks with loops (Pearl, 1988) involve non-local computations and the backpropagation algorithm cannot be applied to them. Diéz (1996) has proposed a technique called *local conditioning* that uses local computations for propagating beliefs through a network with loops. This technique, although not yet implemented, provides a potential avenue for extending BANNER-PR to handle network with loops.

8.3 Extension to General Nodes

The structure revision algorithm used by BANNER relies on the semantics of Noisy-Or/And nodes in order to locate faults in the theory. The interaction of influences affecting such nodes are structured and compositional in that individual influences combine in well-defined ways to produce an overall influence. This makes it possible to break down the overall influence on a Noisy-Or/And node into contributions due to each individual influence, and thus localize faults. It is not clear how these techniques could be extended to cover nodes which allow for arbitrary interactions between individual influences.

The more general idea of using abduction to generate explanations of failures in prediction should be extendible to networks with other models of interactions of influences. One approach is to use two inference passes per misclassified example in order to locate faults in the theory; one inference pass that propagates beliefs through the network based on the given evidence, and a second inference pass that propagates beliefs based on the desired values of the class variables. The first pass determines the beliefs of the nodes in the network given the evidence. The second pass determines what the beliefs of the nodes in the network should be in order to produce the desired outcome. Nodes in the network whose beliefs vary significantly between the two passes indicate discrepancies between the current state of the network and the desired state of the network, and are potential sites for revision. This procedure is similar to the use of abduction in logical theory refinement approaches such as EITHER (Ourston & Mooney, 1994) and NEITHER (Baffes & Mooney, 1993) to determine potential revisions. While such a procedure can identify faulty nodes, it cannot localize faults to individual links in the network. Although adding or deleting links from these faulty nodes would involve search, focusing attention on a small of subset of nodes would still be beneficial by reducing the space of revisions to be considered.

8.4 Integrating other Bayesian Techniques

Our structure revision algorithm proceeds in two phases: one phase that selects specific nodes to be revised, and a second phase that applies a set of revision operators that expand or shrink the parent set of the node to be revised, or add new hidden nodes to the network. As discussed in Chapter 5, BANNER uses the information gain metric (Mitchell, 1997; Quinlan, 1990, 1986; Mingers, 1989) to select the nodes to be added to, or deleted from, the parent set of the node being revised. It would be interesting to study the effects of using the Bayesian scoring metrics employed by other Bayesian network learning techniques (Cooper & Herskovits, 1992; Ramoni & Sebastiani, 1997; Friedman, 1997) for this purpose. These metrics would first have to adapted to Noisy-Or/And nodes. In addition to being more computationally expensive than the information gain metric, they are not geared toward optimizing conditional distributions. However, since they are well integrated into the framework of Bayesian networks, they may be better at selecting appropriate additions and deletions to the network. This is an empirical question that would be interesting to explore.

8.5 Extension to Multi-valued Variables

While all our algorithms have assumed that the variables in the network are binary-valued, several real-world applications of Bayesian networks require multi-valued variables. The notion of Noisy-Or nodes has been extended to handle multi-valued variables (Pradhan et al., 1994). Noisy-and nodes can be similarly extended. We believe that the parameter revision component can be adapted to this case easily. Adapting the theory refinement algorithm poses a challenge, since we now have to search for hidden variables with an unknown number of values, which would increase the search space considerably. Connolly (1993) proposes a technique for discovering hidden variables with an unknown number of values using a clustering algorithm similar to COBWEB (Fisher, 1987). However, their approach can only learn tree-structured networks with all the observable variables at the bottom-most level. It would interesting to explore the possibility of using such techniques to extend BANNER to learn multi-valued variables.

Chapter 9

Conclusion

Theory refinement is an area of research that is based on the idea that, when only limited data is available, biasing an inductive learner with prior knowledge can improve learning by focusing the search space of possible hypotheses. Theory refinement systems assume that the learner has an initial imperfect knowledge base (usually obtained from a domain expert), which is then inductively revised to fit the data. Experiments on real-world data have demonstrated that revising an approximate domain theory produces more accurate results than learning from training data alone. Early research in theory refinement focused on purely symbolic representations, which are very comprehensible, but show poor accuracies on several real-world learning problems. Techniques that revise rule-bases by mapping them into representations that combine logical rules and some form of uncertain reasoning, or numerical summing of evidence, have been shown to result in dramatically improved accuracies. One hybrid representation that is extensively used for this purpose is a class of multi-layered feedforward neural networks, called *Knowledge-Based neural networks*, where the structure of the networks are determined by the initial logical theory (Towell & Shavlik, 1994). Mahoney and Mooney (1993) have proposed a theory refinement technique that uses the *certainty factor* representation (Buchanan & Shortliffe, 1984). However, such hybrid representations lack well-defined semantics and cannot be easily understood.

Bayesian networks, on the other hand, are attractive as hybrid representations because they combine sound mechanisms for representing probabilities with a well-founded qualitative representation of the correlations between variables. The structure of a Bayesian network represents qualitative relationships between variables, and the parameters of the network represent quantitative relationships. Both the structure and the parameters of a Bayesian network have simple semantics that are grounded in probability theory, and several sound inference mechanisms have been developed to reason with such networks.

In the recent years, there has been a growing interest in the problem of learning Bayesian networks from data. Research in this field has made rapid progress in recent years, and many difficult problems, such as learning from incomplete data, have been addressed. However, until now, there has not been a solution to the problem of learning or revising Bayesian networks with hidden variables, shown to be efficient, effective, and scalable through evaluation on real problems. In addition, most Bayesian network learning algorithms are focused on the task of modeling the distribution of the data, without regard to the specific task for which the network is being built. There has been little effort to design algorithms that are specifically geared towards optimizing the classification accuracy of the learned network.

The goal of our research has been two-fold: to develop an efficient and effective approach to revising Bayesian network classifiers with hidden variables, and to develop a hybrid theory refinement technique that revises logical theories by mapping them into Bayesian networks. Unlike the previous approaches to revising Bayesian networks, we were specifically interested in developing mechanisms, similar to those employed by logical theory refinement techniques (Ourston & Mooney, 1994; Koppel et al., 1994; Cohen, 1992; Wogulis & Pazzani, 1993; Richards & Mooney, 1995; Brunk, 1996), for using the data to focus the search for effective modifications to the network.

With this in view, we have developed a technique, implemented in a system called Banner, for revising Bayesian networks with Noisy-Or and Noisy-And nodes. We focused our attention on Noisy-Or/And nodes because their specification requires only a linear number of parameters, affording efficiency with respect to representation, reasoning, and learning. In addition, they are semantically close to logical disjunction and conjunctions, and thus facilitate the use of knowledge expressed in the form of logical rules as an initial theory.

The inputs to Banner are an approximately correct initial theory, either in the form of a Bayesian network with Noisy-Or/And nodes, or in the form of logical Horn-clause rules, and some data consisting of labeled examples that may not include observations of all the variables in the network. An initial theory specified in the form of a logical rules is converted into a Bayesian network with Noisy-Or/And nodes. The goal of Banner is to use the data to improve the classification accuracy of the network by revising its parameters and structure and adding hidden variables when necessary. It uses a two-tiered approach, where it first tries to fit the data by revising the parameters of the network. If this is not sufficient to accurately classify all the examples in the data, the structure of the network is modified in order to correct the misclassifications. The parameter revision and structure revision steps are repeated until convergence. This two-tiered approach makes Banner modular in that it is possible to plug in different algorithms for either of these two phases without affecting the other.

The parameter revision phase assumes that the underlying Bayesian structure is correct, and is concerned with modifying the parameters of the network to improve predictive accuracy. Our implementation of Banner provides two different parameter revision techniques: Banner-Pr and a variant of APN (Russell et al., 1995) calledC-APN. Banner-Pr uses a gradient descent backpropagation algorithm similar to that used in multi-layered feedforward networks. It has been designed specifically to take advantage of the computational efficiency offered by Noisy-Or and Noisy-And nodes, and is targeted towards learning classifiers by optimizing the conditional distribution of the class variables given the evidence. However, it can only be applied to polytrees or virtual polytrees, and for tasks involving causal predictions.

C-APN, based on APN, (Section 2.5) is a more general technique for revising the parameters of a Bayesian network which also uses gradient descent. Whereas APN learns to optimize the likelihood of the entire data being generated by the network, C-APN learns a network that is optimized to estimate the probability of certain class variables given some evidence. Our experiments show that, when applicable, Banner-Pr converges much faster than APN or C-APN, and learns more accurate networks.

As mentioned earlier, the structure revision phase is invoked when it is found that revising the parameters of the network was not sufficient to classify all the examples in the data correctly.

Like most techniques for revising logical theories, our approach to structure revision uses abduction to attribute failures in prediction to specific portions of the network. By augmenting a network with *leak* nodes, our technique uses Bayesian inference to detect specific nodes and links in the network that would have to be revised in order to correct misclassifications. This restricts the search space considerably by focusing attention on a subset of variables that can be held responsible for an erroneous prediction by the network.

We have evaluated our technique on the following real-world learning problems: recognising DNA promoters (Noordewier et al., 1991), recognising DNA splice-junctions (Noordewier et al., 1991), learning student models for a C++ tutor (Baffes, 1994), diagnosing brain disorders in human patients (Tuhrim et al., 1991), and classifying chess end-games (Shapiro, 1983, 1987). These experiments showed that the performance of BANNER is comparable to those of hybrid theory refinement systems such as RAPTURE (Mahoney & Mooney, 1993; Mahoney, 1996) and KBANN (Towell & Shavlik, 1994). They also demonstrate that the strategy of starting with an initial, approximate theory gives BANNER an considerable edge over the Naive Bayes algorithm, which is a purely inductive Bayesian network learning algorithm. Our experiments also show that, on some domains, BANNER performs better than Naive Bayes even when it is given no initial theory. In addition, we have evaluated the structure revision component on BANNER by performing experiments on corrupted versions of the theory for recognizing DNA promoters. These experiments demonstrate the effectiveness of our technique in revising such theories so as to significantly improve their accuracies.

The research presented in this dissertation stands at the cross-roads of theory refinement and Bayesian network learning, making contributions to each of these fields. Our research contributions can be summarized as follows:

- From a Bayesian network learning perspective:

  1. We have introduced a novel technique for revising Bayesian networks that can revise networks with hidden variables as well as discover hidden variables when necessary. This technique is specifically aimed at learning classifiers. We have demonstrated, through experiments on real-world data sets, that this approach can efficiently revise large networks and produce highly accurate classifiers.
  2. Whereas previous techniques for revising Bayesian networks searched through the space of all possible revisions, we have introduced novel mechanisms for using the information in the data to guide the search for useful revisions, thus eliminating a large number of irrelevant revisions from consideration.
  3. Since the initial theory given to BANNER may be in the form of propositional Horn-clause rules, our technique also provides a direct mechanism for incorporating knowledge expressed as propositional Horn-clause rules into a Bayesian network. Since many existing knowledge bases are written in the form of rules, and many experts have become comfortable with this formalism, such a mechanism could potentially ease the process of building Bayesian networks.
  4. Although designed for theory refinement, BANNER can also be used as a system for inductively learning Bayesian networks with hidden variables from incomplete data.
  5. We have also introduced a new technique for revising the parameters of a network with Noisy-Or/And nodes that directly exploits the efficiency afforded by these models, and

is targeted towards learning classifiers by trying to optimize the conditional distribution of the class variables given the evidence. We have shown that this technique converges faster and produces more accurate classifiers than an existing algorithm for learning the parameters of a network.

- From a theory refinement perspective:

  1. We have introduced a novel hybrid theory refinement system that bridges the gap between performance and comprehensibility. Our experiments have shown that its performance is comparable to that of other existing hybrid learning systems. At the same time, the networks produced by BANNER are more easily understood within the framework of Bayesian networks.

Appendix A

Derivation of the Gradients for BANNER-PR

A.1 Gradient for a Noisy-Or Node

Since we are concerned with BANNER-PR, we will assume that all evidence is limited to root nodes in the network, that all target variables are sinks in the graph, and that all inference is causal. Under this assumption, the belief of a node N_k representing a target variable, given the evidence is given by:

$$Bel(N_k = F) = \prod_i (1 - c_{ik} Bel(N_i = T)) \tag{A.1}$$

$$Bel(N_k = T) = 1 - \prod_i (1 - c_{ik} Bel(N_i = T)) \tag{A.2}$$

where c_{ik} is the weight on the link from node N_i to node N_k, F represents the values $False$, and T represents the value $True$.

The error function being minimized is given by

$$E[w] = \frac{1}{2} \sum_\mu (\zeta_k^\mu(N_k = T) - Bel_k^\mu(N_k = T))^2 \tag{A.3}$$

where $\zeta_i^\mu(N_k = T)$ is the value of the k-th target variable for pattern μ of the data.

In order to achieve gradient descent, we require that

$$\begin{aligned} \Delta c_{ij} &\propto -\frac{\partial E[w]}{\partial c_{ij}} && \text{(A.4)} \\ &= -\eta \frac{\partial E[w]}{\partial c_{ij}} && \text{(A.5)} \end{aligned}$$

where η is the constant of proportionality. Since $E[w]$ is summed over all examples in the training set, the contribution to Δc_{ij} from a single example μ is given by:

$$\Delta_\mu c_{ij} = -\eta \sum_k \frac{\partial \frac{1}{2} \sum_k (\zeta^\mu(N_k = T) - Bel^\mu(N_k = T))^2}{\partial c_{ij}} \tag{A.6}$$

$$\Delta^\mu c_{ij} = -\eta \sum_k \frac{\partial \frac{1}{2}(\zeta^\mu(N_k = T) - Bel^\mu(N_k = T))^2}{\partial Bel^\mu(N_j = T)} \times \frac{\partial Bel^\mu(N_j = T)}{\partial c_{ij}} \tag{A.7}$$

where k ranges over all target variables. There are three cases to be considered: one where the node N_j represents a target variable, one where node N_j represents hidden variable directly connected to a target variable, and one where node N_j represents a hidden variable connected indirectly to a target variable. Let us consider these cases separately.

1. **Case 1**: When j is a target variable.

   Since each target variable is a sink, and since all the root nodes have evidence, the right-hand side of Equation A.7 can be written as:

$$\Delta^\mu c_{ij} = \eta(\zeta^\mu(N_j = T) - Bel^\mu(N_j = T)) \times \frac{\partial Bel^\mu(N_j = T)}{\partial c_{ij}} \tag{A.8}$$

   Substituting Equation A.2 into the above, we get

$$\Delta^\mu c_{ij} = \eta(\zeta^\mu(N_j = T) - Bel^\mu(N_j = T)) \times \frac{\partial(1 - \prod_k(1 - c_{kj} Bel^\mu(N_k = T)))}{\partial c_{ij}} \tag{A.9}$$

   Computing the above partial derivatives yields:

$$\Delta^\mu c_{ij} = \eta(\zeta^\mu(N_j = T) - Bel^\mu(N_j = T)) \times \frac{\prod_k(1 - c_{kj} Bel^\mu(N_k = T))}{1 - c_{ij} Bel(N_i = T)} \tag{A.10}$$

   The first term in the equation above is the error associated with the node for pattern μ, δ_j^μ. Thus,

$$\delta_j^\mu = (\zeta^\mu(N_j = T) - Bel^\mu(N_j = T)) \tag{A.11}$$

2. **Case 2**: When node N_j is a hidden variable connected directly to some output variables:

   The right-hand side of Equation A.7 can be written as:

$$\Delta^\mu c_{ij} = -\eta \sum_k \frac{\partial \frac{1}{2}(\zeta^\mu(N_k = T) - (1 - \prod_l(1 - c_{lk} Bel^\mu(N_l = T)))))^2}{\partial Bel^\mu(N_j = T)} \times \frac{\partial Bel^\mu(N_j = T)}{\partial c_{ij}} \tag{A.12}$$

$$\begin{aligned} = \quad & \eta\sum_k(\zeta^\mu(N_k=T)-(1-\prod_l(1-c_{lk}Bel^\mu(N_l=T)))) \\ & \times\frac{\partial(1-\prod_l(1-c_{lk}Bel^\mu(N_l=T)))}{\partial Bel^\mu(N_j=T)} \\ & \times\frac{\partial Bel^\mu(N_j=T)}{\partial c_{ij}} \end{aligned} \tag{A.13}$$

where k ranges over all the target variables, and l ranges over all the parents of the variable N_k. Since

$$(1-\prod_l(1-c_{lk}Bel^\mu(N_l=T)))=Bel^\mu(N_k=T),$$

the last step in the derivation can be re-written as:

$$\begin{aligned} \Delta^\mu c_{ij} \quad = \eta\textstyle\sum_k \quad & (\eta^\mu(N_k=T)-Bel^\mu(N_k=T))\times \\ & \frac{\partial(1-\prod_l(1-c_{lk}Bel^\mu(N_l=T)))}{\partial Bel^\mu(N_j=T)}\times \\ & \frac{\partial Bel^\mu(N_j=T)}{\partial c_{ij}} \end{aligned} \tag{A.14}$$

$$\begin{aligned} = \quad & -\eta\sum_k(\eta^\mu(N_k=T)-Bel^\mu(N_k=T))\times \\ & \frac{\partial\prod_l(1-c_{lk}Bel^\mu(N_l=T))}{\partial Bel^\mu(N_j=T)}\times \\ & \frac{\partial Bel^\mu(N_j=T)}{\partial c_{ij}} \end{aligned} \tag{A.15}$$

$$\begin{aligned} = \quad & \eta\sum_k(\eta^\mu(N_k=T)-Bel^\mu(N_k=T))c_{jk}\times \\ & \prod_{l\neq j}(1-c_{lk}Bel^\mu(N_l=T))\times \\ & \frac{\partial Bel^\mu(N_j=T)}{\partial c_{ij}} \end{aligned} \tag{A.16}$$

$$\begin{aligned} = \quad & \eta\sum_k(\eta^\mu(N_k=T)-Bel^\mu(N_k=T))c_{jk}\times \\ & \prod_{l\neq j}(1-c_{lk}Bel^\mu(N_l=T))\times \\ & \frac{\partial(1-\prod_m(1-c_{mj}Bel^\mu(N_m=T)))}{\partial c_{ij}} \end{aligned} \tag{A.17}$$

where m ranges over the parents of node N_j. The second term is similar to Case 1 above, and computing partial derivative yields:

$$\begin{aligned} \Delta^\mu c_{ij} \quad = -\eta\textstyle\sum_k \quad & (\eta^\mu(N_k=T)-Bel^\mu(N_k=T))c_{jk}\times \\ & \prod_{l\neq j}(1-c_{lk}Bel^\mu(N_l=T))\times \\ & \frac{\partial\prod_m(1-c_{mj}Bel^\mu(N_m=T))}{\partial c_{ij}} \end{aligned} \tag{A.18}$$

$$= \eta\textstyle\sum_k \quad (\eta^\mu(N_k=T)-Bel^\mu(N_k=T))c_{jk}\times$$

$$\prod_{l \neq j} (1 - c_{lk} Bel^{\mu}(N_l = T)) \times$$
$$Bel^{\mu}(N_i = T) \times$$
$$\prod_{m \neq i} (1 - c_{mj} Bel^{\mu}(N_m = T)) \tag{A.19}$$

The first term in the equation above is the error, δ_j^{μ} attributed to node N_j for pattern μ. Thus,

$$\delta_j^{\mu} = \sum_k (\eta^{\mu}(N_k = T) - Bel^{\mu}(N_k = T)) c_{jk} \prod_{l \neq j} (1 - c_{lk} Bel^{\mu}(N_l = T)) \tag{A.20}$$

3. **Case 3**: When node N_j is a hidden variable not directly connected to an output variables, it can be shown that

$$\Delta^{\mu} c_{ij} = \eta \delta_i^{\mu} Bel^{\mu}(N_i = T) \prod_{m \neq i} (1 - c_{mj} Bel^{\mu}(N_m = T)) \tag{A.21}$$

where m ranges over the parents of node N_j, and

$$\delta_i^{\mu} = \sum_j \delta_j^{\mu} \prod_{l \neq j} (1 - c_{lj} Bel^{\mu}(N_l = T)) \tag{A.22}$$

where j ranges over the children of node N_i, l ranges over the parents of node N_j.

The gradient for a Noisy-And node is similarly derived.

Appendix B

Initial Domain Theories

Initial Domain Theories

B.1 DNA Promoter Recognition

B.1.1 Input Features

```
P-50, ..., P-1, P1, ..., P7:   (A G C T)
```

B.1.2 Categories

```
Promoter:          (present absent)
```

B.1.3 Domain Theory

```
(promoter ?x) <- (contact ?x) (conformation ?x)

(contact ?x)  <- (minus_35 ?x) (minus_10 ?x)

(minus_35 ?x) <- (p-37 c) (p-36 t) (p-35 t) (p-34 g) (p-33 a)
                 (p-32 c)
(minus_35 ?x) <- (p-36 t) (p-35 t) (p-34 g) (p-32 c) (p-31 a)
(minus_35 ?x) <- (p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)
                          (p-31 a)
(minus_35 ?x) <- (p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)

(minus_10 ?x) <- (p-14 t) (p-13 a) (p-12 t) (p-11 a) (p-10 a)
                          (p-9 t)
(minus_10 ?x) <- (p-13 t) (p-12 a) (p-10 a) (p-8 t))
(minus_10 ?x) <- (p-13 t) (p-12 a) (p-11 t) (p-10 a) (p-9 a)
                          (p-8 t)
(minus_10 ?x) <- (p-12 t) (p-11 a) (p-7 t)
```

```
(conformation ?x) <- (p-47 c) (p-46 a) (p-45 a) (p-43 t) (p-42 t)
                     (p-40 a) (p-39 c) (p-22 g) (p-18 t) (p-16 c)
                     (p-8 g) (p-7 c) (p-6 g) (p-5 c)
                     (p-4 c) (p-2 c) (p-1 c)
(conformation ?x) <- (p-45 a) (p-44 a) (p-41 a)
(conformation ?x) <- (p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t)
                     (p-16 t) (p-15 g) (p-1 a)
(conformation ?x) <- (p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t)
                     (p-21 a) (p-20 a) (p-17 t) (p-15 t) (p-4 t)
```

B.2 Splice Junction Recognition

B.2.1 Input Features

```
         P-30, ..., P-1, P1, ..., P30:  (A G C T)
```

B.2.2 Categories

```
        IE:                   (present absent)
        EI:                   (present absent)
```

B.2.3 Domain Theory

```
% Prolog notation is used with two minor extensions.
% ::- indicates a "definitional" rule. That is, rules with this
% indicator should never be changed.  These rules merely recode
% inputs into low-level derived features.
% The second extension allows n-of-m style rules. That is rules
% are allowed to say if any 'n' of the following 'm' antecedents
% are true, then the consequent should be considered true.

% Also, the rules use a shorthand notation for expressing sequences.
% Namely, the rule:
%       EI-stop ::- @-3 `TAA'.
% could be expanded to:
%       EI-stop ::- @-3=T, @-2=A, @-1=A.
% In this shorthand, there is no position 0.

% An exon->intron boundary is defined by a short sequence arround
% the boundary and the absence of a "stop" codon on the exon side
% of the boundary.
```

```
EI :- @-3 'MAGGTRAGT', not(EI-stop).

EI-stop ::- @-3 'TAA'.
EI-stop ::- @-3 'TAG'.
EI-stop ::- @-3 'TGA'.
EI-stop ::- @-4 'TAA'.
EI-stop ::- @-4 'TAG'.
EI-stop ::- @-4 'TGA'.
EI-stop ::- @-5 'TAA'.
EI-stop ::- @-5 'TAG'.
EI-stop ::- @-5 'TGA'.

% An intro->exon boundary is defined by a short sequence arround the
% boundary, the absence of a "stop" codon immediately following the
% boundary and a "pryamidine rich" region preceeding the boundary.

IE :- pyramidine-rich, @-3 'YAGG', not(IE-stop).

pyramidine-rich :- 6 of (@-15 'YYYYYYYYYY').

IE-stop1 ::- @1 'TAA'.
IE-stop2 ::- @1 'TAG'.
IE-stop3 ::- @1 'TGA'.
IE-stop4 ::- @2 'TAA'.
IE-stop5 ::- @2 'TAG'.
IE-stop6 ::- @2 'TGA'.
IE-stop7 ::- @3 'TAA'.
IE-stop8 ::- @3 'TAG'.
IE-stop9 ::- @3 'TGA'.

% In addition to the above rules, the following iterative constructs
% can be used as needed to define letters other than {A G C T}.
% These letters represent disjunctive combinations of the four
% nucleotides. The codes are standard in the biological literature.

For i from ((-30 to -1) and (+1 to +30))
           {@<i>'Y' ::- @<i>'C'.
            @<i>'Y' ::- @<i>'T'.}

For i from ((-30 to -1) and (+1 to +30))
           {@<i>'M' ::- @<i>'C'.
            @<i>'M' ::- @<i>'A'.}
```

```
For i from ((-30 to -1) and (+1 to +30))
          {@<i>‘R’ ::- @<i>‘A’.
           @<i>‘R’ ::- @<i>‘G’.}
```

B.3 C++ Tutor

B.3.1 Input Features

```
         pointer:              (constant non-constant absent)
         integer:              (constant non-constant)
         pointer-init:         (true false)
         integer-init:         (true false)
         pointer-set:          (true false)
         integer-set:          (yes no through-pointer)
         multiple-operands:    (true false)
         poistion-A:           (normal left-lazy right-lazy)
         operator-A-lazy:      (AND OR)
         lazy-A-left-value:    (non-zero zero)
         on-operator-A-side:   (left right)
         on-operator-B-side:   (left right)
         operator-A:           (assign modify-assign
                                mathematical logical
                                comparison auto-incr)
         operator-B:           (assign modify-assign
                                mathematical logical
                                comparison auto-incr)
```

B.3.2 Categories

```
         compile-error:        (true false)
         ambiguous:            (true false)
```

B.3.3 Domain Theory

```
compile-error <- constant-not-init
compile-error <- constant-assigned

constant-not-init <- (pointer constant) (pointer-init false)
constant-not-init <- (integer constant) (integer-init false)
```

```
constant-assigned <- (integer constant) integer-init
                     (integer-set yes)
constant-assigned <- (integer-constant) integer-init
                     (integer-set through-pointer)
constant-assigned <- (pointer-constant) pointer-init pointer-set

ambiguous <- multiple-operands operands-linked

operands-linked <- operand-A-uses operator-B-sets
operands-linked <- operand-A-sets operator-B-uses

operand-A-uses <- operand-A-evaluated operator-A-uses

operand-A-sets <- operand-A-evaluated operator-A-sets

operand-A-evaluated <- (position-A normal)
operand-A-evaluated <- (position-A left-lazy)
operand-A-evaluated <- (position-A right-lazy lazy-A-full-eval)

lazy-A-full-eval <- (operator-A-lazy AND)
                    (lazy-A-left-value non-zero)
lazy-A-full-eval <- (operator-A-lazy OR)
                    (lazy-A-left-value zero)

operator-A-uses  <- (on-operator-A-side right)
operator-A-uses  <- (on operator-A-side left)
                    (not (operator-A-assign))

operator-A-sets  <- (operator-A auto-incr)
operator-A-sets  <- (on-operator-A-side left)
                    (operator-A modify-assign)
operator-A-sets  <- (on-operator-A-side left)
                    (operator-A assign)

operator-B-uses  <- (on-operator-B-side right)
operator-B-uses  <- (on-operator-B-side left)
                    (not (operator-B assign))

operator-B-sets  <- (operator-B auto-incr)
operator-B-sets  <- (on-operator-B-side left)
                    (operator-B modify-assign)
operator-B-sets  <- (on-operator-B-side left)
```

```
          (operator-B assign)
```

B.4 Brain Disorders

B.4.1 Input Features

```
decloc-degree:          (0 drowsy stupor coma)
disoriented-degree:     (0 mild moderate severe)
dyspraxia:              (0 present)
hemineglect-side:       (0 right left)
denial:                 (0 present)
compdef-severity:       (0 mild moderate severe)
nonfluency-severity:    (0 mild moderate severe)
repetition-severity:    (0 mild moderate severe)
anomia-severity:        (0 mild moderate severe)
cogabn:                 (0 present)
vf-deficit-side-type:   (0 left-hemianopsia
                         right-hemianopsia
                         left-quadrantanopsia-inferior
                         left-quadrantanopsia-superior
                         right-quadrantanopsia-inferior
                         right-quadrantanopsia-superior)
poorokn-direction:     (0 rhoriz lhoriz vertical)
nystagmus-type:        (0 gazeev horiz-left horiz-right
                        vertical-upbeat vertical-downbeat
                        rotary)
abneom-type:           (0 hgaze-left hgaze-right
                        fourn-right ino-left ino-right
                        vgaze-up vgaze-down thirdn-left
                        thirdn-right skew sixthn-left
                        sixthn-right)
abnpupils-side-type:    (0 left-miosis right-miosis
                         right-mydriasis left-mydriasis)
ptosis-side:           (0 left right)
swallow-severity:      (0 partial unable)
prd-side:              (0 right left)
gag-severity:          (0 impaired absent)
facenumb-side:         (0 left right)
facial-side-type:      (0 right-central left-central
                        right-peripheral left-peripheral)
tongweak-side:         (0 left right)
```

```
dysarthria-severity:    (0 mild moderate severe)
weakness-type:         (0 hemiparesis-right
                        hemiparesis-left monoparesis-lue
                        monoparesis-lle monoparesis-rue
                        monoparesis-rle)
ataxia-type:           (0 limb-right-mild limb-left-mild
                        limb-left-severe limb-right-severe
                        truncal)
decram-side:           (0 right-mild right-severe left-mild
                        left-severe)
abndtrs-side:          (0 right-incdtr right-decdtr
                        left-incdtr left-decdtr)
babs-side:             (0 left right)
gait-type:             (0 lhemi rhemi unsteady other)
dss-side:              (0 left right)
pp-side:               (0 right-mild left-mild
                        right-moderate left-moderate
                        right-severe left-severe)
touch-side:            (0 left right)
temp-side:             (0 left right)
posloss-side:          (0 right left)
vibloss-side:          (0 left right)
twopoint-side:         (0 left right)
agraph-side:           (0 right left)
```

B.4.2 Categories

```
left-internal-capsule:   (present absent)
left-temporal-lobe:      (present absent)
left-frontal-lobe:       (present absent)
left-parietal-lobe:       (present absent)
```

B.4.3 Domain Theory

```
(facenumb-side right) <- (left-internal-capsule present)
(facenumb-side right) <- (left-parietal-lobe present)

(facial-side-type right-central) <- (left-internal-capsule present)
(facial-side-type right-central) <- (left-frontal-lobe present)
```

```
(tongweak-side right) <- (left-internal-capsule present)
(tongweak-side right) <- (left-frontal-lobe present)

(swallow-severity partial) <- (left-internal-capsule present)
(swallow-severity partial) <- (left-frontal-lobe present)
(swallow-severity partial) <- (left-parietal-lobe present)
(swallow-severity partial) <- (left-temporal-lobe present)

(swallow-severity unable) <- (left-internal-capsule present)
(swallow-severity unable) <- (left-frontal-lobe present)
(swallow-severity unable) <- (left-parietal-lobe present)
(swallow-severity unable) <- (left-temporal-lobe present)

(gag-severity impaired) <- (left-internal-capsule present)
(gag-severity impaired) <- (left-frontal-lobe present)
(gag-severity impaired) <- (left-parietal-lobe present)
(gag-severity impaired) <- (left-temporal-lobe present)

(gag-severity absent) <- (left-internal-capsule present)
(gag-severity absent) <- (left-frontal-lobe present)
(gag-severity absent) <- (left-parietal-lobe present)
(gag-severity absent) <- (left-temporal-lobe present)

(dysarthria-severity mild) <- (left-internal-capsule present)
(dysarthria-severity mild) <- (left-frontal-lobe present)
(dysarthria-severity mild) <- (left-parietal-lobe present)

(dysarthria-severity moderate) <- (left-internal-capsule present)
(dysarthria-severity moderate) <- (left-frontal-lobe present)
(dysarthria-severity moderate) <- (left-parietal-lobe present)

(dysarthria-severity severe) <- (left-internal-capsule present)
(dysarthria-severity severe) <- (left-frontal-lobe present)
(dysarthria-severity severe) <- (left-parietal-lobe present)

(weakness-type hemiparesis-right) <- (left-internal-capsule present)
(weakness-type hemiparesis-right) <- (left-frontal-lobe present)

(weakness-type monoparesis-rue) <- (left-internal-capsule present)

(weakness-type monoparesis-rle) <- (left-internal-capsule present)

(decram-side right-mild) <- (left-internal-capsule present)
```

```
(decram-side right-mild) <- (left-frontal-lobe present)

(decram-side right-severe) <- (left-internal-capsule present)
(decram-side right-severe) <- (left-frontal-lobe present)

(babs-side right) <- (left-internal-capsule present)
(babs-side right) <- (left-frontal-lobe present)

(abndtrs-side right-incdtr) <- (left-internal-capsule present)
(abndtrs-side right-incdtr) <- (left-frontal-lobe present)

(abndtrs-side right-decdtr) <- (left-internal-capsule present)
(abndtrs-side right-decdtr) <- (left-frontal-lobe present)

(gait-type rhemi) <- (left-internal-capsule present)
(gait-type rhemi) <- (left-frontal-lobe present)
(gait-type rhemi) <- (left-parietal-lobe present)

(gait-type other) <- (left-internal-capsule present)
(gait-type other) <- (left-frontal-lobe present)
(gait-type other) <- (left-parietal-lobe present)

(dss-side right) <- (left-internal-capsule present)
(dss-side right) <- (left-parietal-lobe present)

(pp-side right-mild) <- (left-internal-capsule present)
(pp-side right-mild) <- (left-parietal-lobe present)

(pp-side right-moderate) <- (left-internal-capsule present)
(pp-side right-moderate) <- (left-parietal-lobe present)

(pp-side right-severe) <- (left-internal-capsule present)
(pp-side right-severe) <- (left-parietal-lobe present)

(touch-side right) <- (left-internal-capsule present)

(temp-side right) <- (left-internal-capsule present)

(posloss-side right) <- (left-internal-capsule present)
(posloss-side right) <- (left-parietal-lobe present)

(vibloss-side right) <- (left-internal-capsule present)
(vibloss-side right) <- (left-parietal-lobe present)
```

```
(twopoint-side right) <- (left-internal-capsule present)
(twopoint-side right) <- (left-parietal-lobe present)

(agraph-side right) <- (left-internal-capsule present)
(agraph-side right) <- (left-parietal-lobe present)

(decloc-degree drowsy) <- (left-frontal-lobe present)
(decloc-degree drowsy) <- (left-parietal-lobe present)
(decloc-degree drowsy) <- (left-temporal-lobe present)

(decloc-degree stupor) <- (left-frontal-lobe present)
(decloc-degree stupor) <- (left-parietal-lobe present)
(decloc-degree stupor) <- (left-temporal-lobe present)

(decloc-degree coma) <- (left-frontal-lobe present)
(decloc-degree coma) <- (left-parietal-lobe present)
(decloc-degree coma) <- (left-temporal-lobe present)

(disoriented-degree mild) <- (left-frontal-lobe present)
(disoriented-degree mild) <- (left-parietal-lobe present)
(disoriented-degree mild) <- (left-temporal-lobe present)

(disoriented-degree moderate) <- (left-frontal-lobe present)
(disoriented-degree moderate) <- (left-parietal-lobe present)
(disoriented-degree moderate) <- (left-temporal-lobe present)

(disoriented-degree severe) <- (left-frontal-lobe present)
(disoriented-degree severe) <- (left-parietal-lobe present)
(disoriented-degree severe) <- (left-temporal-lobe present)

(dyspraxia present) <- (left-frontal-lobe present)

(hemineglect-side right) <- (left-frontal-lobe present)
(hemineglect-side right) <- (left-parietal-lobe present)
(hemineglect-side right) <- (left-temporal-lobe present)

(denial present) <- (left-frontal-lobe present)

(compdef-severity mild) <- (left-frontal-lobe present)
(compdef-severity mild) <- (left-parietal-lobe present)
(compdef-severity mild) <- (left-temporal-lobe present)
```

```
(compdef-severity moderate) <- (left-frontal-lobe present)
(compdef-severity moderate) <- (left-parietal-lobe present)
(compdef-severity moderate) <- (left-temporal-lobe present)

(compdef-severity severe) <- (left-frontal-lobe present)
(compdef-severity severe) <- (left-parietal-lobe present)
(compdef-severity severe) <- (left-temporal-lobe present)

(nonfluency-severity mild) <- (left-frontal-lobe present)
(nonfluency-severity mild) <- (left-parietal-lobe present)
(nonfluency-severity mild) <- (left-temporal-lobe present)

(nonfluency-severity moderate) <- (left-frontal-lobe present)
(nonfluency-severity moderate) <- (left-parietal-lobe present)
(nonfluency-severity moderate) <- (left-temporal-lobe present)

(nonfluency-severity severe) <- (left-frontal-lobe present)
(nonfluency-severity severe) <- (left-parietal-lobe present)
(nonfluency-severity severe) <- (left-temporal-lobe present)

(repetition-severity mild) <- (left-frontal-lobe present)
(repetition-severity mild) <- (left-parietal-lobe present)
(repetition-severity mild) <- (left-temporal-lobe present)

(repetition-severity moderate) <- (left-frontal-lobe present)
(repetition-severity moderate) <- (left-parietal-lobe present)
(repetition-severity moderate) <- (left-temporal-lobe present)

(repetition-severity severe) <- (left-frontal-lobe present)
(repetition-severity severe) <- (left-parietal-lobe present)
(repetition-severity severe) <- (left-temporal-lobe present)

(anomia-severity mild) <- (left-frontal-lobe present)
(anomia-severity mild) <- (left-parietal-lobe present)
(anomia-severity mild) <- (left-temporal-lobe present)

(anomia-severity moderate) <- (left-frontal-lobe present)
(anomia-severity moderate) <- (left-parietal-lobe present)
(anomia-severity moderate) <- (left-temporal-lobe present)

(anomia-severity severe) <- (left-frontal-lobe present)
(anomia-severity severe) <- (left-parietal-lobe present)
(anomia-severity severe) <- (left-temporal-lobe present)
```

```
(cogabn present) <- (left-frontal-lobe present)
(cogabn present) <- (left-parietal-lobe present)
(cogabn present) <- (left-temporal-lobe present)

(poorokn-direction rhoriz) <- (left-frontal-lobe present)
(poorokn-direction rhoriz) <- (left-parietal-lobe present)
(poorokn-direction rhoriz) <- (left-temporal-lobe present)

(nystagmus-type horiz-right) <- (left-frontal-lobe present)
(nystagmus-type gazeev) <- (left-parietal-lobe present)

(abneom-type hgaze-right) <- (left-frontal-lobe present)
(abneom-type hgaze-right) <- (left-parietal-lobe present)
(abneom-type hgaze-right) <- (left-temporal-lobe present)

(weakness-type monoparesis-lue) <- (left-frontal-lobe present)
(weakness-type monoparesis-lle) <- (left-frontal-lobe present)

(gait-type unsteady) <- (left-frontal-lobe present)

(ataxia-type limb-right-mild) <- (left-frontal-lobe present)

(vf-deficit-side-type right-hemianopsia) <-
                              (left-parietal-lobe present)
(vf-deficit-side-type right-hemianopsia) <-
                              (left-temporal-lobe present)

(vf-deficit-side-type right-quadrantanopsia-inferior) <-
                              (left-parietal-lobe present)
(vf-deficit-side-type right-quadrantanopsia-superior) <-
                              (left-temporal-lobe present)
```

Bibliography

Alberts, B. (1988). *Mapping and Sequencing the Human Genome.* National Academy Press, Washington, D.C.

Aliferis, C. F., & Cooper, G. F. (1994). An evaluation of an algorithm for inductive learning of Bayesian belief networks using simulated data. In de Mantaras, R. L., & Poole, D. (Eds.), *Proceedings of the Tenth conference on Uncertainty in Artificial Intelligence*, pp. 8–14 Seattle. Morgan Kaufmann.

Baffes, P., & Mooney, R. (1993). Symbolic revision of theories with M-of-N rules. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1135–1140 Chambery, France.

Baffes, P. T. (1994). *Automatic Student Modeling and Bug Library Construction using Theory Refinement.* Ph.D. thesis, University of Texas, Austin, TX.

Baffes, P. T., & Mooney, R. J. (1996). A novel application of theory refinement to student modeling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 403–408 Portland, OR.

Beinlich, I., Suermondt, H., Chavez, R., & Cooper, G. (1989). The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, pp. 247–256 London, England.

Brunk, C. A. (1996). *An Investigation of Knowledge Intensive Approaches to Concept Learning and Theory Refinement.* Ph.D. thesis, University of California, Irvine, CA.

Buchanan, G., & Shortliffe, E. (Eds.). (1984). *Rule-Based Expert Systems:The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Co., Reading, MA.

Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 52–60.

Burnell, L., & Horovitz, E. (1995). Structure and chance: Melding logic and probability for software debugging. *Communications of the Association for Computing Machinery, 38*(3), 31–41.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261–284.

Cohen, W. (1992). Compiling prior knowledge into an explicit bias. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 102–110 Aberdeen, Scotland.

Connolly, D. (1993). Constructing hidden variables in Bayesian networks via conceptual clustering. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 65–72 Amherst, MA.

Cooper, G. (1990). Computational complexity of probabilistic inference using Bayesian belief networks (research note).. *Artificial Intelligence*, *42*, 393–405.

Cooper, G. G., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Cox, P. T., & Pietrzykowski, T. (1987). General diagnosis by abductive inference. In *Proceedings of the 1987 Symposium on Logic Programming*, pp. 183–189.

Craven, M. W. (1996). *Extracting Comprehensible Models from Trained Neural Networks.* Ph.D. thesis, Department of Computer Sciences, University of Wisconsin-Madison.

Craven, M. W., & Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems*, Vol. 8, pp. 24–30 Denver, CO. MIT Press.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–38.

Dempster, A. (1968). A generalization of Bayesian inference. *J. Royal Statist. Soc.*, *30*, 205–247.

Dietterich, T. G. (1998). Statistical tests for comparing supervised learning algorithms. *Neural Computation.* To appear.

Díez, F. J. (1993). Parameter adjustment in Bayes networks. The generalized noisy OR-gate.. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 99–105.

Diéz, F. J. (1996). Local conditioning in Bayesian networks. *Artificial Intelligence*, *87*, 1–20.

Diéz, F. J. (1997). Private communication..

Domingos, P., & Pazzani, M. (1996). Beyond independence: conditions for the optimality of the simple Bayesian classifier. In Saitta, L. (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 105–112. Morgan Kaufmann.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning.* In press.

Feldman, R., Segre, A., & Koppel, M. (1991). Incremental refinement of approximate domain theories. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 500–504 Evanston, IL.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Frean, M. (1990). The Upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, *2*, 198–209.

Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 125–133 Nashville, Tennessee. Morgan Kaufmann Publishers.

Friedman, N., & Goldszmidt, M. (1996). Building classifiers using Bayesian networks. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1277–1284.

Fung, R., & Del Favero, B. (1995). Applying Bayesian networks to information retrieval. *Communications of the Association for Computing Machinery*, *38*(3), 42–48.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE transactions on Pattern Analysis and Machine Intelligence*, *6*, 721–742.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*, 1–58.

Greiner, R., Grove, A. J., & Schuurmans, D. (1997). Learning Bayesian nets that perform well. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence* Providence, RI.

Heckerman, D. (1986). Probabilistic interpretations for Mycin's certainty factors. In Kanal, L. N., & Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence*, pp. 167–196. North Holland, Amsterdam.

Heckerman, D. (1995). A tutorial on learning Bayesian networks. Tech. rep. MSR-TR-95-06, Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052.

Heckerman, D., Geiger, D., & Chikering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 293–301 Seattle, WA.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation.* Addison Wesley.

Holder, L. (1991). *Maintaining the Utility of Learned Knowledge Using Model-Based Control.* Ph.D. thesis, University of Illinois at Urbana-Champaign.

Kohavi, R., Becker, B., & Sommerfield, D. (1997). Improving simple Bayes. In *Proceedings of the European Conference on Machine Learning.*

Kohavi, R., & Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In Saitta, L. (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning.* Morgan Kaufmann.

Koppel, M., Feldman, R., & Segre, A. M. (1994). Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, *1*, 1–50.

Kulikowski, C. A., & Weiss, S. M. (1991). *Computer Systems That Learn - Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann, San Mateo, CA.

Kwoh, C.-K., & Gillies, D. (1996). Using hidden nodes in Bayesian networks. *Artificial Intelligence*, *88*(1-2), 1–38.

Lam, W., & Bacchus, F. (1994a). Learning Bayesian belief networks. An approach based on the MDL principle. *Computational Intelligence*, *10*, 269–293.

Lam, W., & Bacchus, F. (1994b). Using new data to refine a Bayesian network. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 383–390.

Langley, P., & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the Association for Computing Machinery*, *38*(11), 55–64.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, *19*, 191–201.

Mahoney, J. J. (1996). *Combining Symbolic and Connectionist Learning to Revise Certainty-Factor Rule Bases.* Ph.D. thesis, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 96-260.

Mahoney, J. J., & Mooney, R. J. (1993). Combining connectionist and symbolic learning to refine certainty-factor rule-bases. *Connection Science*, *5*, 339–364.

Mahoney, J. J., & Mooney, R. J. (1994). Comparing methods for refining certainty-factor rule bases. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 173–180 New Brunswick, NJ.

McClelland, J. L., & Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* The MIT Press, Cambridge, MA.

Merz, C., Murphy, P. M., & Aha, D. W. (1996). Repository of machine learning databases `http://www.ics.uci.edu/~mlearn/mlrepository.html`. Department of Information and Computer Science, University of California, Irvine, CA.

Michalski, R. S., & Chilausky, S. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Journal of Policy Analysis and Information Systems*, *4*(2), 126–161.

Mingers, J. (1989). An empirical comparison of selestion measures for decision-tree induction. *Machine Learning*, *3*, 319–342.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, New York, NY.

Mooney, R. J. (1995). A preliminary PAC analysis of theory revision. In Petsche, T., Hanson, S., & Shavlik, J. (Eds.), *Computational Learning Theory and Natural Learning Systems, Vol. 3*, pp. 43–53. MIT Press, Cambridge, MA.

Mooney, R. J. (1997). Integrating abduction and induction in machine learning. In *Working Notes of the IJCAI-97 Workshop on Abduction and Induction in AI.*

Musick, R. C. (1994). *Belief Network Induction.* Ph.D. thesis, University of California at Berkeley.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, *56*, 71–113.

Noordewier, M. O., Towell, G. G., & Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, Vol. 3 San Mateo, CA. Morgan Kaufman.

O'Neill, M., & Chiafari, F. (1989). Escherichia coli promoters. *Journal of Biological Chemistry*, *264*, 5531–5534.

Opitz, D. W. (1995a). An anytime approach to connectionist theory refinement: Refining the topologies of knowledge-based neural network. Tech. rep. 1281, University of Wisconsin-Madison.

Opitz, D. W. (1995b). *An Anytime Approach to Connectionist Theory Refinement: Refining the Topologies of Knowledge-Based Neural Network.* Ph.D. thesis, University of Wisconsin-Madison.

Opitz, D. W., & Shavlik, J. W. (1993). Heuristically expanding knowledge-based neural networks. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 512–517 Chamberry, France.

Ourston, D. (1991). *Using Explanation-Based and Empirical Methods in Theory Revision.* Ph.D. thesis, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 91-164.

Ourston, D., & Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 815–820 Detroit, MI.

Ourston, D., & Mooney, R. (1991). Improving shared rules in multiple category domain theories. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 534–538 Evanston, IL.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, *66*, 311–344.

Pazzani, M., & Brunk, C. (1993). Finding accurate frontiers: A knowledge-intensive approach to relational learning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 328–334 Washington, D.C.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, Inc., San Mateo,CA.

Pradhan, M., Provan, G., Middleton, B., & Henrion, M. (1994). Knowledge engineering for large belief networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 484–490 Seattle, WA.

Provan, G. M., & Singh, M. (1994). Learning Bayesian networks using feature selection. In *Proceedings of the Workshop on Artificial Intelligence and Statistics*, pp. 291–300. Springer-Verlag, New York, Inc.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo,CA.

Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Ramachandran, S., & Mooney, R. J. (1996a). Revising Bayesian network parameters using back-propagation. Unpublished.

Ramachandran, S., & Mooney, R. J. (1996b). Revising Bayesian networks parameters using back-propagation. In *International Conference on Neural Networks: Plenary, Panel and Special Sessions*, pp. 82–87 Washington D.C., USA.

Ramoni, M., & Sebastiani, P. (1997). Learning bayesian networks from incomplete databases. In Geiger, D., & Shenoy, P. (Eds.), *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence.* Morgan Kaufmann Publishers, Inc.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *13*, 81–132.

Richards, B. L., & Mooney, R. J. (1995). Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, *19*(2), 95–131.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*, 465–471.

Rumelhart, D., Durbin, D., Golden, R., & Chauvin, Y. (1995). Backpropagation: The basic theory. In Chauvin, W., & Rumelhart, D. (Eds.), *Backpropagation: Theory, Architectures, and Applications*, pp. 1–34. Lawrence Erlbaum Associates, Hillsdale, NJ.

Russell, S., Binder, J., Koller, D., & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1146–1152 Montreal, Canada.

Schwalb, E. (1993). Compiling Bayesian networks into neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 291–297 Amherst, MA.

Shafer, G. (1976). *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, NJ.

Shapiro, A. D. (1983). *The Role of Structured Induction in Expert Systems.* Ph.D. thesis, University of Edinburgh.

Shapiro, A. D. (1987). *Structured Induction in Expert Systems.* Addison-Wesley.

Shortliffe, E., & Buchanan, B. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, *23*, 351–379.

Siegel, A. F. (1988). *Statistics and data analysis: An Introduction*, chap. 15, pp. 336–339. John Wiley and Sons.

Singh, M. (1997). Learning Bayesian networks from incomplete data. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 534–539 Providence, Rhode Island.

Singh, M., & Provan, G. (1996). Effective learning of selective Bayesian network classifiers. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 453–461 Bari, Italy.

Spiegelhalter, D. J., & Lauritzen, S. L. (1990). Sequential updating of conditional probabilities on directed graphical structures. *Networks*, *20*, 579–605.

Srinivas, S., & Breese, J. (1993). Ideal: Influence diagram evaluation and analysis in Lisp: Documentation and users' guide. Tech. rep. No. 23, Rockwell International Science Center, Palo Alto: Rockwell.

Thiesson, B. (1995). Accelerated quantification of Bayesian networks with incomplete data. In Fayyad, U. M., & Uthurusamy, R. (Eds.), *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 306–11. AAAI Press.

Thompson, C. A. (1993). Inductive learning for abductive diagnosis. Tech. rep. Masters Thesis, Department of Computer Sciences, University of Texas, Austin, TX.

Thompson, C. A., & Mooney, R. J. (1994). Inductive learning for abductive diagnosis. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 664–669 Seattle, WA.

Thompson, K., Langley, P., & Iba, W. (1991). Using background knowledge in concept formation. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 554–558 Evanston, IL.

Towell, G. G. (1991). *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* Ph.D. thesis, University of Wisconsin, Madison, WI.

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, *70*, 119–165.

Towell, G. G., Shavlik, J. W., & Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 861–866 Boston, MA.

Towell, G., & Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, *13*(1), 71–102.

Tuhrim, S., Reggia, J., & Goodall, S. (1991). An experimental study of criteria for hypothesis plausibility. *Journal of Experimental and Theoretical Artificial Intelligence*, *3*, 129–144.

Watson, J., Roberts, H., Steitz, J., & Weiner, A. (1987). *The Molecular Biology of Gene*. Benjamin-Cummings, Menlo Park, CA.

Weigand, A., Huberman, B., & Rumelhart, D. (1990). Predicting the future: A conenctionist approach. *International Journal of Neural Systems*, *I*, 193–209.

Wogulis, J. (1991). Revising relational domain theories. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 462–466 Evanston, IL.

Wogulis, J. (1994). *An Approach to Repairing and Evaluating First-Order Theories Containing Multiple Concepts and Negation*. Ph.D. thesis, University of California, Irvine, CA.

Wogulis, J., & Pazzani, M. (1993). A methodology for evaluating theory revision systems: Results with Audrey II. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1128–1134 Chambery, France.

Zadeh, L. A. (1981). Possibility theory and soft data analysis. In Cobb, L., & Thrall, R. M. (Eds.), *Mathematical Frontier of Social and Policy Sciences*, pp. 69–129. Westview.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.

Vita

Sowmya Ramachandran was born in Madras, India, on November 11, 1965, the daughter of Shantha Ayyar and Dr. R. R. Ayyar. She graduated from Chinmaya Vidyalya in 1981, and entered the Indian Institute of Technology, Madras, where she earned the degree of Bachelor of Technology in Computer Science. She subsequently moved to Bombay, India, to work for Tata Consultancy Services. She enrolled at Rutgers, The State University of New Jersey, New Brunswick, NJ, in August 1989, where she received the degree of Master of Science in Computer Science. Funded by a Microelectronics and Computer Development (MCD) Fellowship, she joined the doctoral program in the Department of Computer Sciences at the University of Texas at Austin in August, 1991.

Permanent Address: 14, Cross St.,
Ravi Colony, St. Thomas Mount,
Madras, India, 600016

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1]LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.